



# GRAPH THEORY IN MODERN ENGINEERING

Volume 98

Ernest J. Henley &  
R. A. Williams

# **GRAPH THEORY IN MODERN ENGINEERING**

This is Volume 98 in  
MATHEMATICS IN SCIENCE AND ENGINEERING  
A series of monographs and textbooks  
Edited by RICHARD BELLMAN, *University of Southern California*

The complete listing of books in this series is available from the Publisher upon request.

# GRAPH THEORY IN MODERN ENGINEERING

*Computer Aided Design, Control, Optimization,  
Reliability Analysis*

ERNEST J. HENLEY and R. A. WILLIAMS

*Cullen College of Engineering  
University of Houston  
Houston, Texas*



ACADEMIC PRESS New York San Francisco London 1973

*A Subsidiary of Harcourt Brace Jovanovich, Publishers*

**COPYRIGHT © 1973, BY ACADEMIC PRESS, INC.**

**ALL RIGHTS RESERVED.**

**NO PART OF THIS PUBLICATION MAY BE REPRODUCED OR  
TRANSMITTED IN ANY FORM OR BY ANY MEANS, ELECTRONIC  
OR MECHANICAL, INCLUDING PHOTOCOPY, RECORDING, OR ANY  
INFORMATION STORAGE AND RETRIEVAL SYSTEM, WITHOUT  
PERMISSION IN WRITING FROM THE PUBLISHER.**

**ACADEMIC PRESS, INC.**

**111 Fifth Avenue, New York, New York 10003**

*United Kingdom Edition published by*

**ACADEMIC PRESS, INC. (LONDON) LTD.**

**24/28 Oval Road, London NW1**

**LIBRARY OF CONGRESS CATALOG CARD NUMBER: 72-77353**

**PRINTED IN THE UNITED STATES OF AMERICA**

# CONTENTS

<b>PREFACE</b> . . . . .	ix
<b>ACKNOWLEDGMENTS</b> . . . . .	xii
<b>NOMENCLATURE</b> . . . . .	xiii

## Chapter 1. **Basic Concepts**

Directed Graphs . . . . .	1
Flow Graphs for Differential Equations . . . . .	4
Further Terminology . . . . .	6

## Chapter 2. **Construction and Analysis of Signal Flow Graphs**

Signal-Flow-Graph Reduction and the Solution of Equations. .	9
Flow-Graph Algebra . . . . .	10
Loop Elimination and Equation Solving by Equation Reduction	12
Mason's Rule . . . . .	15
Applying Mason's Rule . . . . .	16
Application of Mason's Rule to Dynamic Systems . . . . .	19
References . . . . .	21

## Chapter 3. **Systematic Analysis of Large Flow Graphs**

Determining Specific Paths and Loops . . . . .	24
Determining Specific Paths . . . . .	31
List Processing Approach to Determining Paths . . . . .	33
Evaluation of Gains . . . . .	35

Removing Unnecessary Paths from the List Structure . . . . .	36
Evaluating Determinants in Mason's Formula . . . . .	37
Computing the Determinants Corresponding to Specific Paths as Required by Mason's Rule . . . . .	40
Computing Transmittances . . . . .	40
Implementing the Algorithms . . . . .	41
References . . . . .	43
 <b>Chapter 4. Frequency Response Analysis</b>	
Final Value Theorem . . . . .	45
Initial Value Theorem . . . . .	46
Substitution Rule . . . . .	47
Bode Diagrams . . . . .	48
A Typical Application . . . . .	50
Other Stability Criteria . . . . .	51
Reference . . . . .	52
 <b>Chapter 5. Sensitivity Analysis</b>	
Sensitivity Analysis in General . . . . .	53
Derivation of Network Functions . . . . .	56
Deriving Sensitivity Functions . . . . .	59
Applying Sensitivity Functions to Structural Analysis . . . . .	60
Gain and Phase Sensitivity . . . . .	63
Pole and Zero Sensitivity . . . . .	64
Sensitivities via Structural Methods . . . . .	65
Sensitivities for Large Parameter Variations . . . . .	68
Transfer Functions and Sensitivities in Static Systems . . . . .	68
References . . . . .	69
 <b>Chapter 6. Examples and Further Applications</b>	
Closed-Form Solution of an Absorption Column . . . . .	71
Sensitivity Analysis of a Heat Exchanger Network . . . . .	78
Continuous Stirred Tank Reactor . . . . .	83
Eigenvalue Problems . . . . .	100
Ordering Recycle Calculations . . . . .	102
Transportation Problems . . . . .	105
References . . . . .	107
 <b>Chapter 7. Linear and Nonlinear Programming</b>	
General Form of LP Problems . . . . .	109
Simplex Solution by Signal-Flow-Graph Methods . . . . .	112
Example Problems. . . . .	113

Discussion of Solution Procedure (MASNLP) . . . . .	118
An Advantage of MASNLP . . . . .	118
Dual Simplex Method . . . . .	120
Postoptimal Analysis . . . . .	121
Sensitivity Analysis . . . . .	122
Nonlinear Systems . . . . .	122
Method of Feasible Directions . . . . .	125
A Heat Exchanger Problem . . . . .	131
Solution to the Heat Exchanger System . . . . .	133
Results and Discussion . . . . .	139
References . . . . .	148

**Chapter 8. Ranking by Flow-Graph Methods**

Tennis Rankings . . . . .	150
Implementation Difficulties . . . . .	153
A Solution to the Implementation Problem . . . . .	158
Reference . . . . .	159

**Chapter 9. Undirected Graphs**

Undirected Graphs . . . . .	161
Trees . . . . .	162
Logic Trees . . . . .	163
Branch and Bound Methods . . . . .	167
Trees—Further Theorems and Definitions . . . . .	169
Disconnecting Sets and Cut-Sets . . . . .	171
Bipartite Graphs . . . . .	171
Assigning Output Sets . . . . .	173
References . . . . .	174

**Chapter 10. Matrix Representation of Graphs**

Nondirected Graphs . . . . .	177
Digraphs . . . . .	179
References . . . . .	188

**Chapter 11. Branch and Bound, Search Tree Methods**

Formal Definition of the Branch and Bound Technique . . . . .	190
The Traveling Salesman Problem . . . . .	192
Integer Programming and Pseudo-Boolean Programming . . . . .	205
Generalization of the Branch and Bound Method . . . . .	209
Bidding Contest Problem . . . . .	209
Conclusion . . . . .	220
References . . . . .	221



Chapter 12. **Process Reliability Analysis by Flow-Graph Methods**

Module Representation of Reliability Graphs . . . . . 224

Principle of Path Enumeration and Sensitivity Calculation Methods 226

Basic Algorithm . . . . . 228

Path Finding Algorithm . . . . . 229

System Reliability . . . . . 230

Comparison with State Enumeration Algorithm . . . . . 231

Examples . . . . . 235

Extension to System MTBF Calculation . . . . . 242

Conclusions . . . . . 243

References . . . . . 244

Appendix A. **Matrix Theory** . . . . . 245

Appendix B. **Linearization of Equations** . . . . . 253

Appendix C. **Derivation of Mason’s Rule** . . . . . 257

Appendix D. **Boolean and Modulo-2 Algebra** . . . . . 265

Appendix E. **Linear Programming** . . . . . 269

Appendix F. **The Coates Flow Graph** . . . . . 277

Appendix G. **The Fibonacci Search Method** . . . . . 281

Appendix H. **Glossary of Graph Nomenclature** . . . . . 285

Appendix I. **Computer Programs for Chapters 1–6** . . . . . 289

Appendix J. **Computer Programs for Chapter 7** . . . . . 293

BIBLIOGRAPHY . . . . . 295

INDEX . . . . . 297

## PREFACE

The Swiss mathematician Euler is universally credited with having produced the first paper on graph theory. By examining the node-vertex relationships in a graph representing the seven bridges, two islands, and two banks of the Prengel River, he solved the “Koenigsberg Bridge Puzzle” which posed the problem of whether it is possible to begin at any of the four land areas, walk across each bridge exactly once, and return to the starting point.

Euler’s work raised the curtain on over two hundred years of mathematical fun-and-games with graphs. The literature is extensive and fascinating. Professor F. Harary prefaces the bibliography of his recent book “Graph Theory” with an appropriate quotation from Lewis Carroll: “and thick and fast they came at last, and more, and more, and more.” There are literally dozens of named topological theorems involving intersections, combinations, factorizations, and enumeration of graphs. The theory is rich and the applications are varied. Disregarding the puzzle-oriented papers such as “Calculation of the Maximum Number of Snakes in a Box,” a partial and random listing of titles from Harary’s extensive bibliography is an indication of the breadth of the applications:

“Valence-Isomerism of Cyclopolyenes”

“A Graph Theory Approach to a Communications Problem”

“Facilities Planning with Graph Theory”

“Space-Time Approach to Quantum Electrodynamics”

“Theory of Games and Economic Behavior”

“On the Theory of the Virial Development of the Equation of State of Monatomic Gases”

“A Theorem on Graphs with an Application to a Problem in Traffic Control”

“Linear Graphs and Electrical Network”

“Bi-Path Networks and Multicommodity Flows”

“Successive Approximation Methods in Classical Statistical Mechanics”

Thus, in addition to being a natural tool in the theory of mathematical relations, we see potentially useful work in network flow analysis, optimization communications, theoretical chemistry, economics, and control.

Of increasing importance is the application of graph theory to problems in engineering design and analysis. The major credit for organizing flow graphs for this purpose must be accorded S. J. Mason. By organizing graphs so that the independent variables represent excitations, and the nodes and paths dependent variables and functional relations respectively, Mason obtained graphs of equations representative of many fields of engineering. Furthermore, Mason systematized the topological transformations along with their attending algebraic transformation, thus showing how one can use a graph to obtain relationships between input excitations and output variables.

The usefulness of Mason’s work to engineers can be greatly enhanced by coupling it with two additional mathematical tools: matrix theory and combinatorial algebra. Graphs, consisting as they do of points and lines, can also be represented by matrices whose elements are the connections between the points. In many instances it is easier to manipulate matrices than graphs. Combinatorial mathematics enters the framework naturally, since paths and loops are nothing more than combinations of the individual edges of a graph.

Although we have attempted here to provide a fairly balanced coverage of the theory and applications of matrix- and graph-manipulation methods, if we as authors can be considered as having a bias, it is in the direction of the combinatorial approach to flow-graph analysis. It is our experience that these methods are considerably more versatile and powerful than either graph reduction or matrix methods. In view of the extensive algebra involved, combinatorial analysis of large engineering systems must be done by computer. It is for this reason that a series of computer programs developed by the authors are highlighted in this book.<sup>†</sup> Their application to the analysis of control systems, linear and nonlinear optimization,

<sup>†</sup> All programs described in this book are available from the authors.

sensitivity analysis, ranking problems, formulation of design algorithms, structural analysis and synthesis, and transportation problems constitute the major thrust of this book.

Since our interest lies in the application of flow graphs to engineering problems, we discuss in this book only those elements of graph theory necessary for an understanding of the applications. In view of the many excellent topologically oriented books available, it would be presumptuous of us to intrude unnecessarily into this field. Our focus is mostly on “directed” graphs in which the direction of movement of the signal is known. However, since undirected graphs in general and tree structures in particular are becoming increasingly important in engineering, some coverage is included.

It was the members of Professor D. Wildes’ research group at Stanford, where one of us (EJH) was privileged to spend the summer of 1971 as a Visiting Professor, who convinced us that branch and bound and similar graph concepts will play an increasingly dominant role in design and analysis.

## ACKNOWLEDGMENTS

Chapter 11, "Branch and Bound, Search Tree Methods" was contributed by Dr. Jean-Paul Barthes of Stanford. All of the other new material was developed at the University of Houston by graduate students and post-doctoral researchers. Chapters 3, 4, 5, and 6 constitute the bulk of R. A. Williams' doctoral thesis in chemical engineering. Chapter 7 is taken from the M. S. dissertation of Mr. Takashi Tonomura, and Chapter 12 represents the combined efforts of Mr. Satish Gandhi, a doctoral student, and Professor Koichi Inoue of the University of Kyoto, who was a visiting professor at Houston from 1971-1973.

It is a pleasure to acknowledge the many helpful suggestions offered by colleagues and friends at the University of Houston; J. Bailey, R. Flumerfelt, C. Donaghey, D. Luss and, most particularly, R. L. Motard. Mrs. Judy Stover who accurately and noncomplainingly typed, and retyped, and retyped this manuscript most certainly qualifies for an author's best-friend-of-the-year award. Without the encouragement of Dean C. V. Kirkpatrick and A. E. Dukler, Chairman of the Chemical Engineering Department at Houston this book could not have been produced.

A special note of thanks is due to the Office of Naval Research whose support under ONR Contract N0014-68-A-0151 made this book possible and to Professors F. Harary of the University of Michigan and K. Inoue of the University of Kyoto who provided extremely valuable and incisive reviews. The material in Chapter 12 was developed with the help of a National Science Foundation Grant CK-34542.

## NOMENCLATURE

$a$	Used along with $b$ , $c$ , $d$ , etc. to designate items, nodes, or branch gains
$a_{ij}$	Element in row $i$ , column $j$ , of a matrix $A$
$A$	Matrix of coefficients for a system of equations
$A$	Tank area; Heat exchanger area; Constant
$A_k$	Constant defined in Eq. (12.5)
$A(s)$	Network function, see Fig. 5.1
$AR$	Amplitude ratio
$B$	Vector of "right-hand sides" for a system of equations
$B$	Bandwidth; Constant
$B, B'$	Bids on a contractors-lots problem
$B(s)$	Network function, see Fig. 5.1
$c$	Heat capacity of cold stream; Profit coefficient in integer and pseudo-Boolean programming
$c$	Row vector in circuit matrix
$C$	Concentration; Restrictive condition; Heat capacity of hot stream; Constant
$C_N$	Circuit matrix, nondirected graph
$C'$	See Eq. (10.22)
$C(s)$	Feedback network function, see Fig. 5.1
$d$	Discount in bidding contest
$D(s)$	Denominator of a transfer function
$D(s, p)$	Denominator of a transfer function, stresses dependence on parameter $p$
$D_1(s)$	See Eq. (5.15)
$D_2(s)$	See Equation (5.15)
$e$	Edge; Current edge during description of algorithms
$E$	Error
$E$	Matrix in revised simplex method
$f$	Damping coefficient; Function; Force
$F$	Degrees of freedom

$\mathbf{F}$	Vector of forces
$g$	Constraint function; Evaluation function
$G$	Gain; Gas flow rate; Graph
$G_c$	Coates graph
$G_{co}$	Subgraph of a Coates graph
$HO$	Temperatures (see Fig. 7.8)
$I$	Input node
$\mathbf{I}$	Identity matrix
$\mathbf{I}_B'$	See Eq. (10.21)
$\mathbf{I}_C'$	See Eq. (10.21)
$\mathbf{I}_D$	Incidence matrix, digraph
$\mathbf{I}_N$	Incidence matrix, nondirected graph
$j$	Imaginary number $\sqrt{-1}$
$J$	Objective function
$k$	Chemical reaction rate constant; Number of lots in bidding contest; Hooke's constant; Number of trees
$K$	Constant
$K_c$	Controller gain
$K_n$	Complete graph with $n$ vertices
$l$	Lower bound in algorithms; Number of lots in bidding contest
$L$	Loop gain; Liquid flow rate
$L_k$	Loops in the one-connection in a Coates graph
$L_l$	Loops in the $l$ th connection of a Coates graph
$L^0$	Upper bound in algorithms
$m$	Henry's constant; Mass
$M$	Manipulated variable; Total variables; Number of minimal paths
$\bar{M}$	Mean time between failures
$n$	Total number of items
$N$	Total number of items (or equations); Total number of lots in bidding contest
$N^*$	Set of nonnegative integers
$N_i$	Number of units in the $i$ th module
$N(s)$	Numerator of a transfer function
$N(s, p)$	Numerator of a transfer function, stresses dependence on parameter $p$
$N_1(s)$	See Eq. (5.15)
$N_2(s)$	See Eq. (5.15)
OR	Logical "or" operation
$p$	A particular parameter; Number of lots in bidding contest; Pole of a transfer function
$P$	Path gain
$\bar{P}$	Problem which is considered
$\bar{P}_{co}$	Connection gain in Coates graph
$P_i$	Minimal path
$Pr$	Probability
$q$	Represents $f(s, p)$
$Q$	Flow rate
$\mathbf{r}$	Directional vector in MFD
$r_i$	$i$ th root of a polynomial; Reliability of the identical unit in the $i$ th mode
$r_{ij}$	Reliability of the $j$ th unit in the $i$ th mode

$r^0$	Reliability of a unit in service
$r^1$	Reliability of a unit in standby
$R$	Return difference; Cold stream outlet temperatures; Total system reliability
$\mathbf{R}$	Adjacency matrix
$R_i$	Reliability of the $i$ th mode
$\mathbf{R}_r$	Reachability matrix
$s$	Laplace variable
$\mathbf{s}$	Row vector in cut-set matrix
$S$	Cardinality of $S$ , i.e., number of elements in $S$ ; Cold stream inlet temperature; Set of possible solutions to a problem
$S'$	See Eq. (10.23)
$\mathbf{S}_D$	Cut-set matrix directed graph
$\mathbf{S}_N$	Cut-set matrix, nondirected graph
$S_{N_i}$	System sensitivity with respect to the number of units in the $i$ th module
$S_{r_{ij}}$	System sensitivity with respect to the unit $r_{ij}$
$S_{R_i}$	System sensitivity with respect to the module $R_i$
$S_{0lm}^{(i)} \dots pq$	Node of a search tree at the $i$ th level
$S_p^T$	Logarithmic sensitivity of transmittance or transfer function $T$ with respect to parameter $p$
$t$	Time; Temperature of cold stream
$T$	Transmittance or transfer function; Time constant; Temperature of a hot stream
$\mathbf{T}$	Transition matrix
$T(s)$	Transfer function
$T(s, p)$	Transfer function, stresses dependence on parameter $p$
$T_0(s)$	Leakage transmission
$u(t)$	Forcing function
$U$	Heat transfer coefficient
$\mathbf{U}$	Vector of sensitivities
$UG$	Combination heat transfer coefficient and geometrical factor
$v$	Edge; Velocity
$v(t)$	Velocity as a function of time
$V$	Volume
$\mathbf{V}$	Vector of velocities
$w_{ij}$	Element in row $i$ , column $j$ , of the $\mathbf{W}$ matrix
$w_c$	Heat flow rate, cold stream
$WC$	Heat flow rate, hot stream
$x$	Output, dependent variable; Concentration in liquid phase
$x(t)$	Output as a function of time
$\mathbf{X}$	Vector of dependent variables; Eigenvector
$y$	Concentration in the vapor phase
$Z$	Objective function; Number of paths and path unions
$\alpha$	Gain function; Grouping of parameters
$\beta$	Phase function; Grouping of parameters
$\Delta$	System determinant, delta; Finite difference
$\theta$	Deviation variable
$\lambda$	Eigenvalue; Ratio of number of lots; Incremental length in gradient method



$\lambda_i$	Constant failure rate of the identical unit in the $i$ th module
$\lambda_{ij}$	Constant failure rate of the $j$ th unit in the $i$ th module
$\lambda_{oi}$	Constant failure rate in service of the identical unit in the $i$ th module
$\lambda_{is}$	Constant failure rate in standby of the identical unit in the $i$ th module
$\pi$	Partition of nodes
$\rho$	Density
$\sigma$	Solution to the branch and bound problem
$\tau$	Time constant; Dead time
$\phi$	Phase angle; Symbol for the empty set
$\gamma$	Circuit rank
$\omega$	Angular frequency
$\omega_r$	Resonant frequency
$\Omega$	Elementwise Boolean multiplication

### Subscripts

$A$	Component $A$
$B$	Component $B$ ; Branch
$B_{set}$	Set point
$C$	Chord
$D$	Derivative action
$i, j, k$	Designates $i$ th, $j$ th, and $k$ th elements
$i \rightarrow j$	Designates "from $i$ to $j$ "
$I$	Integral action
$p$	Parameter $p$
$0$	Initial, entry, leakage or steady state condition
$0lm \dots pq$	Path from a node of a search tree to the root
$\infty$	Ultimate condition

### Superscripts

$(i)$	Level of a node in the search tree
$0$	Null condition
$T$	Transmittance or transfer function; Transpose
$T(s, p)$	Transfer function, stresses dependence on parameter $p$
$*$	Optimal value

### Mathematical Symbols

$\cup$	Union (set sum) of two sets
$\cap$	Intersection (set product of two sets)
$\binom{n}{r}$	Binomial coefficient $\frac{n(n-1) \cdots (n-r+1)}{r!}$
$\in$	Included in, part of a set
$\notin$	Not included in, not part of a set

## CHAPTER 1

# BASIC CONCEPTS

This chapter introduces the concepts and nomenclature of directed graphs. Only a very small portion of the hundreds of topological lemmas and theorems applicable to graphs have been useful to engineers, and we restrict our coverage accordingly. We attempt very few proofs of theorems throughout this book, the emphasis is on applications, and the reader is directed to the bibliography at the end of the volume for proofs, and more detailed expositions on graph theory.

### Directed Graphs

*Directed graphs* consist of directed *branches*, interconnected at *nodes*. These nodes are variously called *vertices*, *junction points*, or *points*; the branches are also referred to as *arcs*, *links*, or *edges*, and the graphs themselves are alternately called *signal flow graphs* (SFG), *directed networks*, *digraphs*, or *directed linear graphs*. A graph is said to be connected if it contains no pair of vertices which are not joined by branches. Since we deal almost exclusively with connected graphs in engineering work, unless

stated otherwise, the word connected is always implied when the term graph is used.

The nomenclature is far from standardized, and, with a burgeoning literature originating literally from all parts of the world, no standardization can be expected in the foreseeable future. To help the reader ascend this Tower of Babel, a glossary of graph-related terms is included as Appendix H.

The nodes in a graph may represent anything from cities on a map to warehouses, depending on the application. The branches connecting the nodes indicate that the entities represented by the nodes are somehow related. Usually, each branch in the network is assigned a numerical value, a variable, or a function to quantitatively define the relationships between a pair of nodes. The numerical value can be distance, time, or a parameter in an equation; the function may be an integration, or differentiation. In a SFG, where the branches are assigned directions, the relationships between the nodes indicated by the branches is not reciprocal.

Signal flow graphs bear a resemblance to the *block diagrams* used in the theory of servomechanisms. There is, however, a major point of departure. Flow graphs are a graphic representation of sets of linear algebraic or linear differential equations. Each vertex of a graph represents a variable of the equation whereas in a block diagram, the block vertices usually represent a group of physical elements. That flow graph manipulations can be done readily only for a linear system does not restrict their usefulness in engineering analysis unduly. Most of the applications discussed in Chapters 6 and 7 involve nonlinear systems which have been linearized, and the agreement with theory is excellent. Methods of linearization are included as Appendix B.

Figure 1.1 is a SFG representation of the equations

$$\begin{aligned} x_1 &= 5 - 2x_2 & \text{or} & & 5 &= x_1 + 2x_2 + 0x_3 & (1) \\ x_2 &= 5 - x_3 & \text{or} & & 5 &= 0x_1 + x_2 + x_3 & (2) \\ x_3 &= 5x_1 - x_2 & \text{or} & & 0 &= 5x_1 - x_2 - x_3 & (3) \end{aligned} \quad (1.1)$$

The coefficients  $a_{ij}$  of the equations written alongside the branches are referred to as *gains*, *branch gains*, or *transmittances*. They are the operators which map node  $x_i$  into node  $x_j$ , i.e.,  $x_i = a_{ij}x_j$ . The term *transfer function* is frequently used if  $x_i$  and  $x_j$  are functions of the Laplace operator  $s$ . The constants of Eq. (1.1) are the *inputs* or *driving functions* shown in the graph.

The representation of Eq. (1.1) shown in Fig. 1.1 is not unique. Figure 1.2 is a different graph for the same set of equations. In this representation,

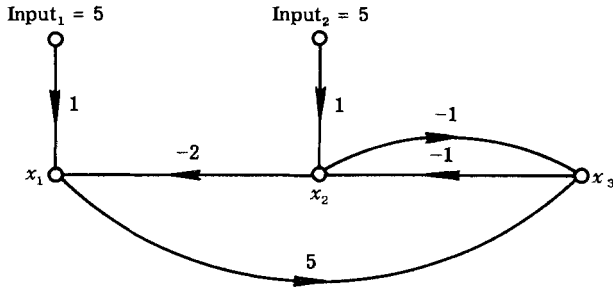


Fig. 1.1. Graphical representation of Eq. (1.1).

the variables  $x_1$ ,  $x_2$ , and  $x_3$  were obtained from Eqs. (3), (2), and (1), respectively, and two dummy input nodes,  $I_1$  and  $I_2$ , and a dummy output node,  $x_3 = x_4$ , have been inserted. An *input node*, *source node*, or *initial node*, is defined as a node with only outgoing branches, and an *output node*, *terminal vertex*, or *sink* is a node with only incoming branches.  $x_4$  is an output node;  $I$  is an input node. Some authors use the term output node synonymously with variable of interest ( $x_1$ ,  $x_2$ ,  $x_3$  in this example).

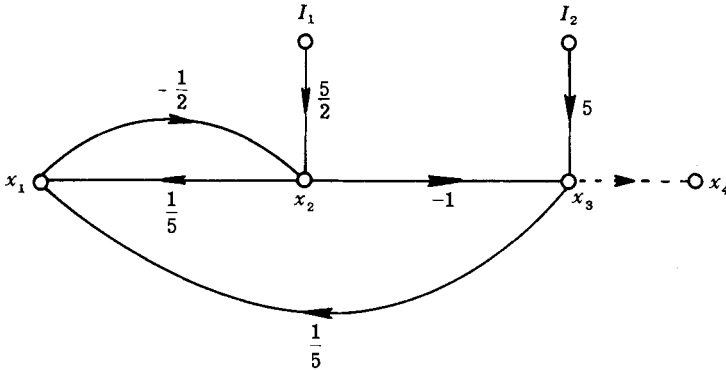


Fig. 1.2. Graphical representation of Eq. (1.1).

Matrix representations of equations are convenient manipulative and expository tools. Equation (1.1) can, for example, be written as<sup>†</sup>

$$\begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 1 \\ 5 & -1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \\ 0 \end{bmatrix} \quad (1.2)$$

<sup>†</sup> Appendix A gives a summary of matrix theory and linear algebra.

This suggests that graphs, like the equations they spring from, can be given equivalent matrix representations. A detailed discussion of this is deferred until Chapter 10.

### Flow Graphs for Differential Equations

Signal flow graphs can be used to represent dynamic systems which are describable by sets of linear or linearized differential equations. To illustrate the method by which the SFG for a dynamic system is derived, the classical model of one wheel of an automobile suspension system will be considered. The system consists of a mass  $m$  supported by a spring having a Hooke's constant  $k$ , and a shock absorber having a damping coefficient  $f$ . The differential equations which mathematically model the system are

$$m \, dv/dt + fv + kx = u(t) \quad (1.3)$$

$$dx/dt = v \quad (1.4)$$

In the above equations,  $x$  represents the displacement of the mass  $m$  from a fixed reference point, while  $t$  and  $v$  represent time and velocity, respectively.

There are four variables of interest,  $dv/dt$ ,  $v$ ,  $dx/dt$ , and  $x$ . We have, however, only two equations in four unknowns, so we must write two additional relationships. These are the integrations which produce  $x$  and  $v$  from  $dx/dt$  and  $dv/dt$ .

$$x = \int_{x(0)} \frac{dx}{dt} dt = \int_0^t \frac{dx}{dt} dt + x(0) \quad (1.5)$$

$$v = \int_{v(0)} \frac{dv}{dt} dt = \int_0^t \frac{dv}{dt} dt + v(0) \quad (1.6)$$

A graph corresponding to these four equations is drawn by defining a node for each variable and each input, and connecting these with branches having the required gains. In Fig. 1.3 the nodes are inputs  $v(0)$ ,  $x(0)$ ,  $u(t)$ , and the variables  $v$  and  $x$  and their time derivatives. Again, alternate representations are possible.

It is unusual to find graphs such as Fig. 1.3 in the modern literature. The almost universal practice is to work in the  $s$  domain with Laplace-transformed equations. The conventional procedure for constructing graphs is to solve the transformed equations for their highest order derivative prior

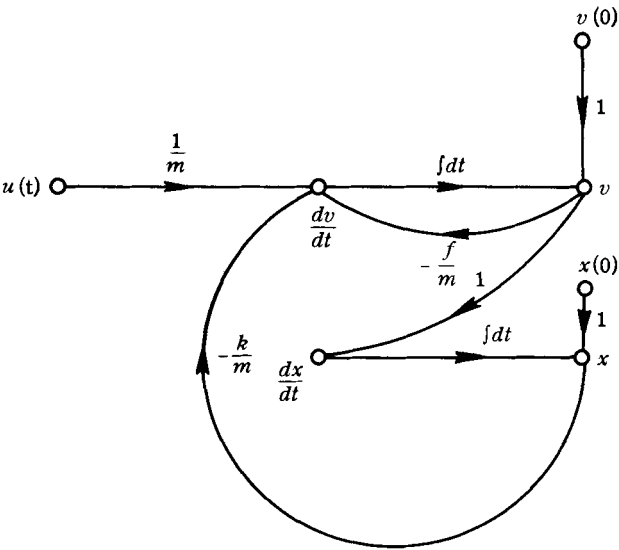


Fig. 1.3. SFG representation of Eqs. (1.3) and (1.4).

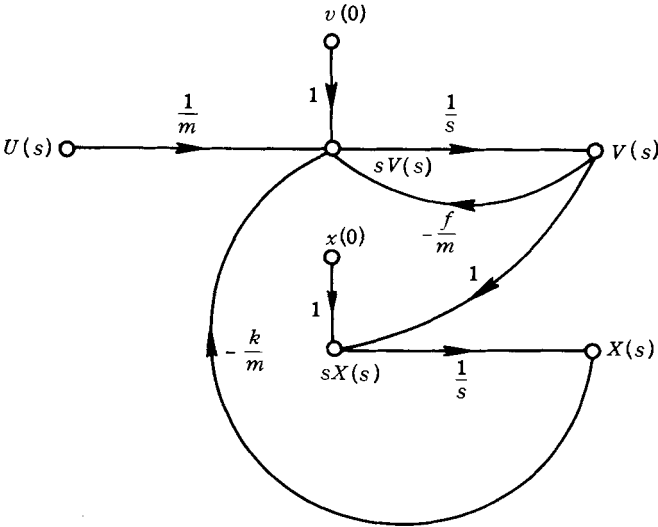


Fig. 1.4. SFG representation of Eqs. (1.7) and (1.8).

to making the graph. For Eqs. (1.3) and (1.4)

$$sV(s) = (1/m)U(s) - (f/m)V(s) - (k/m)X(s) + v(0) \quad (1.7)$$

$$sX(s) = V(s) + x(0) \quad (1.8)$$

The integrations are achieved by inserting branches from  $sV(s)$  to  $V(s)$ , and from  $sX(s)$  to  $X(s)$ , each branch having a gain of  $1/s$ , the Laplace operator for integration. Figure 1.4 is a representation of Eqs. (1.7) and (1.8).

Representations of other dynamic systems can be achieved in an analogous fashion. In general, there will be nodes for each variable and its derivatives, and some of the branch gains will be of the form  $as^n$  where  $a$  represents a function of the system parameters, and  $n$  is a power of the Laplace operator  $s$ .

### Further Terminology

A *path* is defined as a set of at least two connected branches. Paths can be defined either by enumerating the successive branches  $a_{ij}-a_{jk}-a_{kl}-\dots$  or the successive nodes  $x_i-x_j-x_k$ . The product of the branch gains along a path is the *path gain* or *path transmittance*.

Paths can be *open*, in which case no node is crossed more than once. We speak of *forward* or *direct paths*, where the successive nodes are of higher number, and *reverse*, or *inverse paths* where successive nodes are of descending number.

A path originating and terminating on the same node is called a *loop*, *closed path*, *feedback path*, or a *circuit*. Graphs which contain loops are said to be *cyclic*. A *maximal cycle*  $M$  of a graph  $G$  is a cycle such that every other loop in  $G$  is contained in  $M$  or has no vertex in common with it. *Cascade* or *acyclic* graphs are signal flow graphs which have no loops.

## CHAPTER 2

### CONSTRUCTION AND ANALYSIS OF SIGNAL FLOW GRAPHS

When dealing with a set of algebraic equations such as

$$\begin{aligned}a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 &= b_1 \\a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 &= b_2 \\a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 &= b_3 \\a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 &= b_4\end{aligned}\tag{2.1}$$

the  $a$ 's may be regarded as system coefficients, the  $b$ 's as inputs, and the  $x$ 's as outputs. If the equations are solvable, they can be rearranged such that one output variable with a coefficient of unity is separated from each equation. There are many different ways of selecting the output variables; each gives rise to a slightly different flow graph. In the set (2.1) above, let us solve the first for  $x_1$ , the second for  $x_2$ , etc.

$$\begin{aligned}x_1 &= (1/a_{11})(b_1 - a_{12}x_2 - a_{13}x_3 - a_{14}x_4) \\x_2 &= (1/a_{22})(b_2 - a_{21}x_1 - a_{23}x_3 - a_{24}x_4) \\x_3 &= (1/a_{33})(b_3 - a_{31}x_1 - a_{32}x_2 - a_{34}x_4) \\x_4 &= (1/a_{44})(b_4 - a_{41}x_1 - a_{42}x_2 - a_{43}x_3)\end{aligned}\tag{2.2}$$



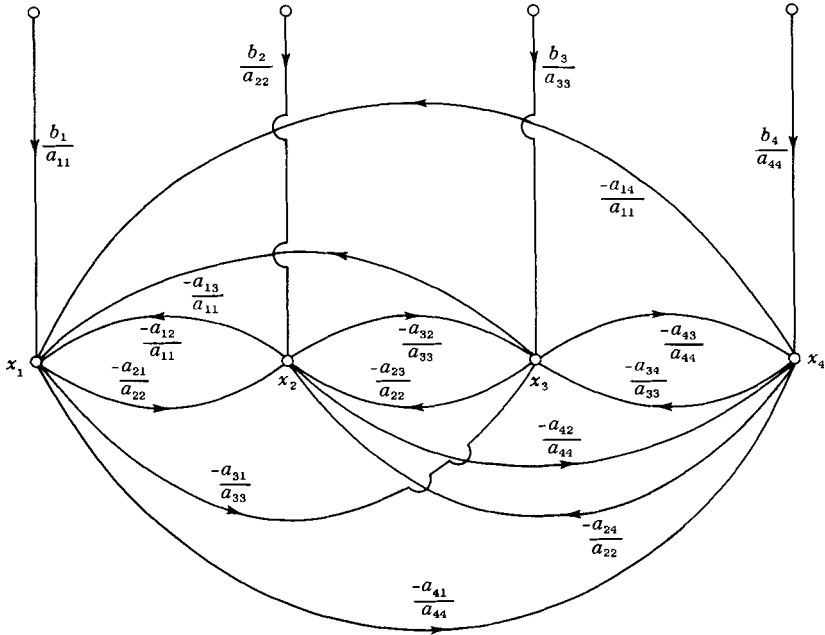


Fig. 2.1. Flow graph of Eq. (2.2).

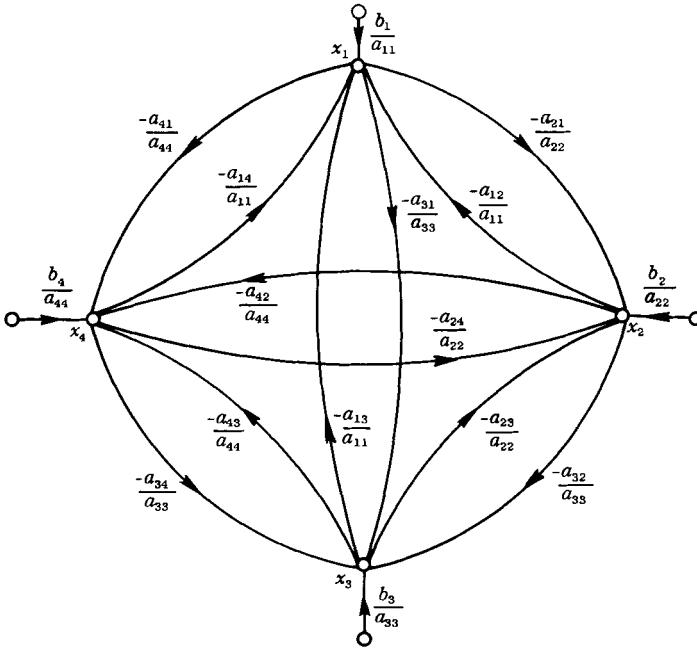


Fig. 2.2. Flow graph of Eq. (2.2).

This type of rearrangement of Eq. (2.1) leads to a Mason signal flow graph. Alternate methods of writing the equations (such as  $\mathbf{Ax} - \mathbf{B} = 0$ ) result in different graphs (a Coates graph in this case). Only Mason graphs appear in Chapters 2–7. The Coates, and other graphs are discussed in Appendix F.

Figures 2.1 and 2.2 show two different arrangements of the Mason graph equivalence of Eq. (2.2).

Although Fig. 2.2 is considerably “neater” than Fig. 2.1, both graphs are cluttered to the point of incomprehension. This, unfortunately, is a major problem in SFG analysis. If the systems are large it is difficult to extract information from graphs. Two potential solutions to this problem are (1) eliminate the graphs and use only their matrix equivalence, or (2) reduce the graphs by combining edges and nodes.

### Signal-Flow-Graph Reduction and the Solution of Equations

Signal-flow-graph reduction is a technique whereby a complex graph is reduced to a simpler, yet equivalent, form called a *residual graph*. The reduction process involves the application of a set of rules by which nodes are absorbed and branches are combined to form new branches having different gains. By repeatedly applying the reduction rules, a complex SFG is reduced to one having the degree of complexity desired. If a graph is completely reduced, the result is a residual graph in which one input node is connected to a selected output node by a single branch. Thus, graph reduction can be viewed as a way of solving sets of linear equations.

The technique of reducing graphs is generally applicable only in the case of SFG's where a limited number of output nodes are of interest. During the reduction process some of the nodes are necessarily absorbed and thereby lose their identity. This is a serious handicap in applications where the preservation of intermediate nodes is of importance. Overcoming this handicap requires that the graph be repeatedly reduced, preserving a set of selected nodes each time.

Programming the graph reduction procedure for implementation on digital computers is not particularly easy, especially in the case of dynamic systems where the branch gains are functions of the Laplace variable  $s$ . These symbolic gains lead to serious computational difficulties associated with the necessary symbol manipulation.

Despite the difficulties involved, graph reduction and symbol manipulation algorithms have been implemented through the use of general purpose

list processing languages such as LISP [1], GRASPE [2], SLIP [3], SNOBOL [4], and COMIT [5]. Network analysis programs are also available [5a].

### Flow-Graph Algebra

Three basic algebraic rules apply:

1. *The addition rule* (Fig. 2.3): the value of a variable designated by a node equals the sum of all gains entering the node

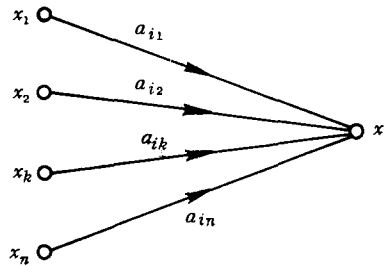
$$x_i = \sum_{j=1}^n a_{ji} x_j \quad (2.3)$$

2. *The transmission rule* (Fig. 2.4): the value of the variable represented by a node is transmitted on every branch leaving the node

$$x_i = a_{ki} x_k, \quad i = 1, 2, \dots, n \quad (2.3a)$$

3. *The product rule* (Fig. 2.5): a cascaded flowgraph is a series connected graph. If it has  $n$  nodes, the  $n - 1$  branches can be replaced with a single branch whose transmittance function equals the product of the old ones.

The removal of nodes to form residual graphs is further illustrated in Fig. 2.6 which demonstrates the graph construction and node elimination process for a cascaded, spring assemblage. An input force  $f_1$  is exerted



Thus

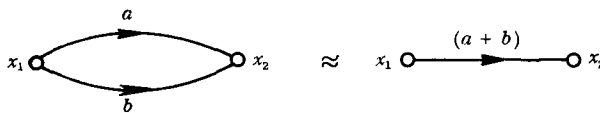
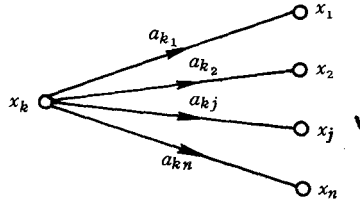


Fig. 2.3. The addition rule.



Thus

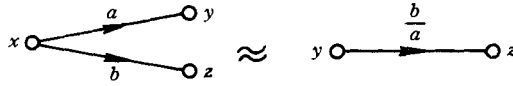
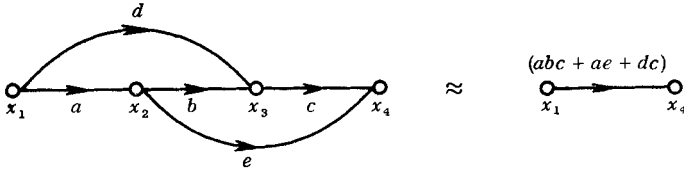


Fig. 2.4. The transmission rule.



and

$$x_1 (abc + ae + dc) = x_4$$

Fig. 2.5. The product rule.

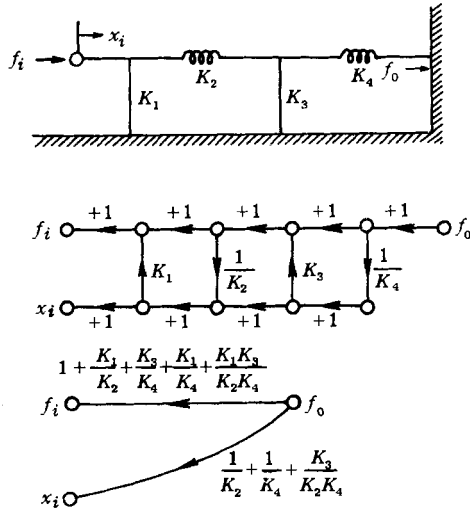
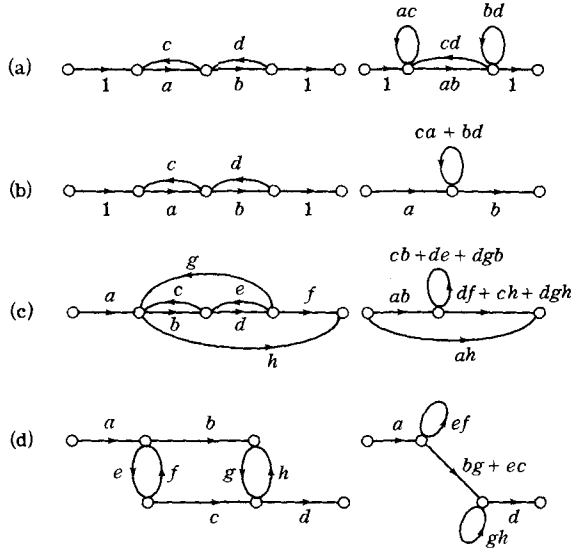


Fig. 2.6. Mechanical-spring assembly and the reduction of a representative cascade flow graph. ( $K_1$  and  $K_3$  are leaf springs.) (From C. S. Lorens, "Flowgraphs," p. 7. McGraw-Hill, New York, 1964.)



**Fig. 2.7.** (a)–(d) Reduction by elementary transformations. (From L. P. A. Robichaud, M. Boisvert, and J. Robert, "Signal Flow Graphs and Applications," p. 11. Prentice-Hall, Englewood Cliffs, New Jersey, 1962.)

against the free end of the assemblage; an output force  $f_o$  results against the wall at the fixed end of the assembly.

Additional examples of node elimination in cyclic graphs by the elementary transformations are shown in Fig. 2.7. These transformations do not necessarily reduce either the complexity or the *index* of the graph, the index being defined as the number of nodes which must be split to remove all loops from a graph.

### Loop Elimination and Equation Solving by Equation Reduction

Systematic ways of reducing flowgraphs based on Gauss–Jordan or other linear equation solving methods have been developed. Implicit in these procedures is the assumption that only a relationship between one input and one output is required.

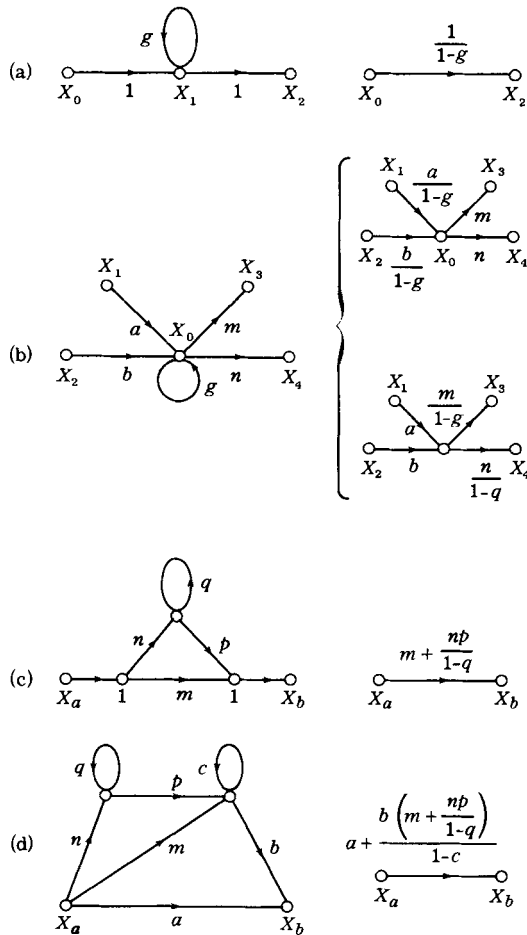
The first step in the process usually consists of removing self loops. This can be done by taking the self-loop (diagonal) terms out of the coefficient matrix. Thus

$$x_j = \sum_{\substack{i=1 \\ i \neq j}}^n a_{ij}x_i + a_{jj}x_j \quad (2.4)$$

becomes

$$x_j = \sum_{\substack{i=1 \\ i \neq j}}^n \frac{a_{ij}}{1 - a_{jj}} x_i \quad (2.5)$$

We observe that to remove a self loop, it is necessary to divide all incoming transmissions by  $(1 - a_{ii})$ . Some examples of loop elimination are shown in Fig. 2.8.



**Fig. 2.8.** Examples of loop elimination. (From L. P. A. Robichaud, M. Boisvert, and J. Robert, "Signal Flow Graphs and Applications," p. 13. Prentice-Hall, Englewood Cliffs, New Jersey, 1962.)

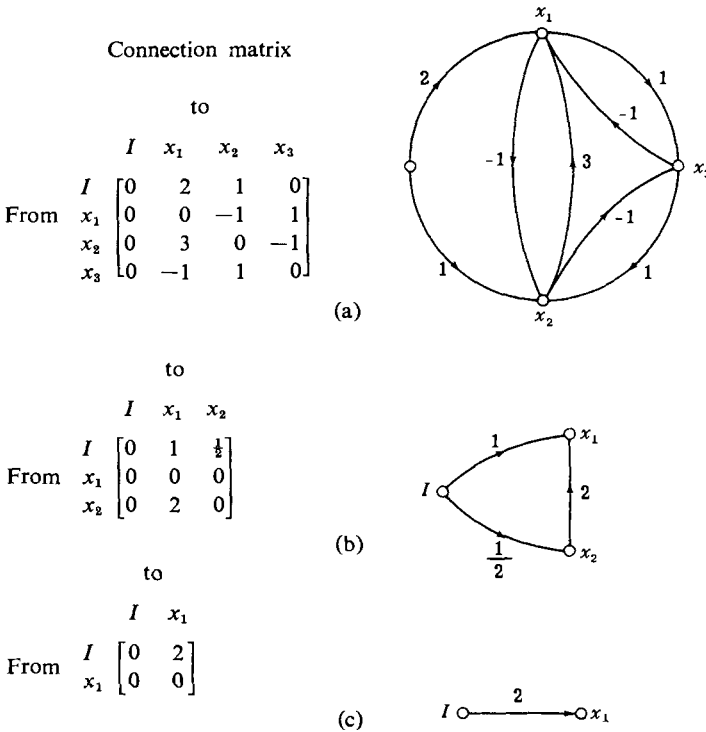
The next step in the procedure is to eliminate  $x_j$  from all other equations by substituting Eq. (2.4) for  $x_j$ . Eliminating  $x_j$  from the equation for  $x_k$ , for example, produces

$$x_k = \sum_{\substack{i=1 \\ i \neq j}}^n \left( a_{ik} + \frac{a_{ij}a_{jk}}{1 - a_{ij}} \right) x_i \quad (2.6)$$

Continued reductions of this type lead to a one-step cascade graph, as will be shown by manipulating the SFG in Fig. 2.9. The flow graph, which will be solved for  $x_1$ , is a graphical representation of the equation

$$x_1 = 2 + 3x_2 - x_3, \quad x_2 = 1 - x_1 - x_3, \quad x_3 = x_1 - x_2 \quad (2.7)$$

As a first step in the reduction, we eliminate node 3. This is equivalent to substituting for  $x_3$  into the equations for  $x_1$  and  $x_2$ .



**Fig. 2.9.** (a) SFG and its connection matrix. (b) Reduction of Fig. 2.9 by elimination of node  $x_3$ . (c) Further reduction of Fig. 2.9 by elimination of node  $x_2$ . The solution is  $x_1 = 2I$ .

Next, the node  $x_2$  is removed. This is equivalent to substituting for  $x_2$  into the equation for  $x_1$ . Thus, we obtain  $x_1 = 2$ , which is the solution to Eq. (2.7).

Although this procedure is relatively simple, and lends itself well to computer algorithms, all information pertaining to the effects of any particular parameter,  $a_{ij}$  on the output  $x_1$  is lost by the time the manipulations are completed. If a solution for  $x_2$  or  $x_3$  is required, the procedure must be repeated.

### Mason's Rule

In SFG language, the value of an unknown in a linear system can be expressed as

$$x_i = \sum_{j=1}^n (T_{j \rightarrow i})(\text{input})_j \quad (2.8)$$

where  $x_i$  is the  $i$ th output,  $n$  is the number of inputs, and  $T_{j \rightarrow i}$  is the transmittance (path gain) from input node  $j$  to output node  $i$ . The *transmittance*  $T_{j \rightarrow i}$  is actually the constrained derivative of  $x_i$  with respect to the  $j$ th input.  $T_{j \rightarrow i}$  can also be thought of as the ratio

$$T_{j \rightarrow i} = \frac{\text{Contribution of input } j \text{ to output } i}{\text{input } j} \quad (2.9)$$

When all of the transmittances are found, the set of equations represented by a graph is essentially solved.

An ingenious technique for evaluating the transmittance  $T$  from a specified input node to an output node in a graph based on the formulation represented by Eq. (2.2) was devised by Mason [6]. The technique is commonly referred to as Mason's rule and is expressed as<sup>†</sup>

$$T = (\sum P_k \Delta_k) / \Delta \quad (2.10)$$

The summation in this equation is over all of the paths leading from an input node to an output node. The quantities  $P_k$  and  $\Delta_k$  are associated with the  $k$ th path.

The quantity  $\Delta$  in Eq. (2.10) has a value equal to the value of the determinant of the system of Eqs. (2.1) and is computed from information pertaining to the loops present in the SFG representation.

<sup>†</sup> A derivation of Mason's rule is given in Appendix C.



$L_k$  is the product of the branch gains encountered in tracing the loop. If any two loops in the signal flow graph have a common node, these loops touch. The expression for  $\Delta$  is formed as follows:

$$\begin{aligned}
 \Delta = & 1.0 - (\sum \text{loop gains}) \\
 & + (\sum \text{product of loop gains taken two at a time but} \\
 & \quad \text{not containing the product of any two loop} \\
 & \quad \text{gains which correspond to touching loops}) \\
 & - (\sum \text{product of loop gains taken three at a time but} \\
 & \quad \text{not containing any terms which contain the} \\
 & \quad \text{product of loop gains corresponding to touch-} \\
 & \quad \text{ing loops}) \\
 & + (\sum \text{product of loop gains taken four at a time} \\
 & \quad \text{with same conditions as previously}) \\
 & - \dots
 \end{aligned} \tag{2.11}$$

The  $\Delta$  for a system with three nontouching loops is

$$\Delta = 1.0 - (L_1 + L_2 + L_3) + (L_1L_2 + L_1L_3 + L_2L_3) - (L_1L_2L_3) \tag{2.12}$$

If loops  $L_1$  and  $L_3$  touch, the corresponding  $\Delta$  is

$$\Delta = 1.0 - (L_1 + L_2 + L_3) + (L_1L_2 + L_2L_3) \tag{2.13}$$

The quantity  $P_k$  in Eq. (2.10) represents the gain of the  $k$ th direct path from the input node of interest to the output node of interest. The quantity  $\Delta_k$  associated with a path  $P_k$  is easily obtained from  $\Delta$  by deleting those terms which contain loop gains corresponding to loops touched by path  $P_k$ .

### Applying Mason's Rule

Considering Eqs. (1.1) and the associated signal flow graph shown in Fig. 1.1, we see that there are two loops:  $x_2 \rightarrow x_3 \rightarrow x_2$  having a loop gain  $L_1$  of  $(-1) \cdot (-1) = (1)$ , and  $x_1 \rightarrow x_3 \rightarrow x_2 \rightarrow x_1$  having a loop gain  $L_2$  of  $(5)(-1)(-2) = (10)$ . Delta is then

$$\begin{aligned}
 \Delta &= 1 - L_1 - L_2 + \cancel{L_1L_2} \\
 &= 1 - 1 - 10 \\
 &= -10
 \end{aligned} \tag{2.14}$$

The term  $L_1L_2$  was deleted from Eq. (2.14) because the two loops touch at nodes  $x_2$  and  $x_3$ .

In evaluating  $x_2$ , for instance, one path is seen to lead from input<sub>1</sub> to  $x_2$ . This path is input<sub>1</sub>  $\rightarrow$   $x_1$   $\rightarrow$   $x_3$   $\rightarrow$   $x_2$  having a gain  $P_1$  of  $(1)(5)(-1) = (-5)$  and a  $\Delta_1$  of

$$\Delta_1 = 1 - \cancel{L_1} - \cancel{L_2} = 1$$

This expression follows directly from Eq. (2.14), the loop gains  $L_1$  and  $L_2$  having been deleted since the path touches both loops. The transmittance from node input<sub>1</sub> to  $x_2$  is now obtained using Mason's rule.

$$T_{\text{input}_1 \rightarrow x_2} = (\sum P_i \Delta_i) / \Delta = [(-5)(1)] / -10 = 0.5$$

The summation in this equation contains only one term since there is only one path from input<sub>1</sub> to  $x_2$ . Also, only one path leads from input<sub>2</sub> to  $x_2$ . This path is input<sub>2</sub>  $\rightarrow$   $x_2$  having a gain  $P_1$  of 1.0 and a  $\Delta_1$  of 1.0, as in the previous case, since the path touches both loops at node  $x_2$ . The transmittance from node input<sub>2</sub> to  $x_2$  is now obtained using Mason's rule.

$$T_{\text{input}_2 \rightarrow x_2} = (\sum P_i \Delta_i) / \Delta = [(1)(1)] / -10 = -0.1$$

Having evaluated all of the necessary transmittances, the value of  $x_2$  is computed via Eq. (2.8).

$$\begin{aligned} x_2 &= \text{input}_1 \cdot T_{\text{input}_1 \rightarrow x_2} + \text{input}_2 \cdot T_{\text{input}_2 \rightarrow x_2} \\ &= (5)(0.5) + (5)(-0.1) \\ &= 2.5 - 0.5 \\ &= 2.0 \end{aligned}$$

This procedure has indeed yielded the correct value for  $x_2$ . In a similar fashion, the value of  $x_1$  and  $x_3$  can be shown to be 1.0 and 3.0, respectively.

As a further demonstration of the use of Mason's rule, we offer Engenstadt's [7] flow graph solution to the heat exchanger control problem investigated by Haskins and Sliepcevich [8]. In the heat exchanger of Fig. 2.10, hot oil is being cooled by a coolant.

The variables are:  $T_{\text{in}}$ ,  $T_{\text{f}}$  = hot oil temperatures in and out;  $T_{\text{cin}}$ ,  $T_{\text{co}}$  = coolant temperature in and out;  $T_{\text{w}}$  = wall temperature;  $W$ ,  $W_{\text{c}}$  = oil and coolant flow rates respectively;  $K$  = gain constant for controller.

The specific problem of interest is to calculate the fluctuations of the outlet oil temperature ( $T_{\text{f}}^*$ ) due to variations in the inlet coolant flow rate, ( $W_{\text{c}}^*$ ) and to find the controller gain which makes  $T_{\text{f}}^*$  independent of

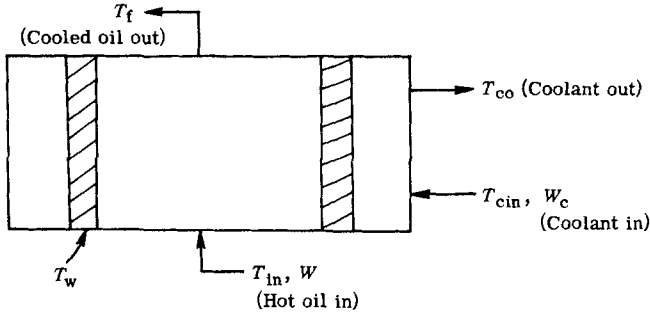


Fig. 2.10. Heat exchange system of Haskins and Sliepcevic.

$W_c^*$ . In flow chart terminology, we require the transmittance  $T_{W_c^* \rightarrow T_f^*}$ . By writing the heat balances in terms of transient ( $T^*$ ) and steady state ( $T_{ss}$ ) temperatures, where  $T = T_{ss} + T^*$ , and, evaluating the constants from system parameters and experimental data, the following linearized steady state equations were obtained by Haskins and Sliepcevic:

$$\begin{aligned}
 0 &= dT_f^*/dt = -136.3T_f^* + 31.7T_w^* + 10.6W^*, \\
 0 &= dT_w^*/dt = -83.3T_w^* + 41.1T_f^* + 21.0T_{co}^*, \\
 0 &= dT_{co}^*/dt = -171T_{co}^* + 82.8T_w^* - 67.6W_c^* \\
 W^* &= KT_w^*
 \end{aligned} \tag{2.15}$$

These equations can be manipulated to form the Mason graph of Fig. 2.11.

We now apply Mason's rule

$$T_{W_c^* \rightarrow T_f^*} = (\sum P_k \Delta_k / \Delta) \tag{2.16}$$

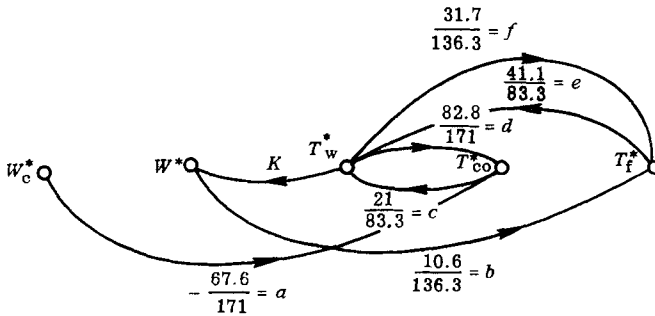


Fig. 2.11. SFG for heat exchanger.

The loops, all of which touch, are

$$L_1 = fe, \quad L_2 = Keb, \quad L_3 = dc$$

Thus

$$\Delta = 1 - (L_1 + L_2 + L_3) = 1 - fe - Keb - dc$$

The paths from the source to sink are

$$P_1 = acf, \quad \Delta_1 = 1 \quad (\text{all loops touch the forward path})$$

$$P_2 = acKb, \quad \Delta_2 = 1 \quad (\text{all loops touch the forward path})$$

Hence

$$T_{W_c^* \rightarrow T_t^*} = \frac{acf + acKb}{1 - fe - Keb - dc} \quad (2.17)$$

To achieve a comparison between this result and that obtained by Haskins and Sliepcevich, we examine the invariance condition where the control is such that the oil outflow temperature  $T_t^*$  is invariant to fluctuations in  $W_c$ . We require  $T_{W_c \rightarrow T_t^*} = 0$ ; thus

$$acf = -acKb \quad (2.18)$$

Solving for  $K$ , we obtain  $K = -2.99$ , which is the correct result.

### **Application of Mason's Rule to Dynamic Systems**

The automobile suspension system considered in Chapter 1 will be used now as an example of the applicability of Mason's rule to dynamic systems. The signal flow graph representation of the system was shown in Fig. 1.4.

The signal flow graph is seen to contain two loops. The first loop is  $sV(s) \rightarrow V(s) \rightarrow sV(s)$ , and has a loop gain  $L_1$  of  $-f/ms$ . The second loop is  $sV(s) \rightarrow V(s) \rightarrow sX(s) \rightarrow X(s) \rightarrow sV(s)$ , and has a loop gain  $L_2$  of  $-k/ms^2$ . The function  $\Delta$  is computed via Eq. (2.11) as

$$\Delta = 1 - L_1 - L_2 + L_1 L_2 \quad (2.19)$$

$$\Delta = 1 + f/ms + k/ms^2 \quad (2.20)$$

To evaluate the output  $X(s)$ , the transmittance from each input node to  $X(s)$  must be evaluated. Considering the input  $U(s)$ , one path having a

gain  $1/ms^2$  is seen to lead from  $U(s)$  to  $X(s)$ . The  $\Delta_1$  associated with this path is just 1.0 since the path touches both loops. The transfer function from input  $U(s)$  to  $X(s)$  is then obtained via Mason's rule as  $(1/ms^2)(1/\Delta)$ . With respect to the second input,  $v(0)$ , there is again but one path leading to  $X(s)$ . This path has a gain of  $1/s^2$  and a  $\Delta_1$  of 1.0 since it also touches both loops. Applying Mason's rule again yields the transfer function from  $v(0)$  to  $X(s)$  as  $(1/s^2)(1/\Delta)$ . The output node,  $X(s)$ , can be accessed from the third input,  $x(0)$ , by a single path which has a gain of  $1/s$ . The  $\Delta_1$  corresponding to this path is  $(1.0 - L_1)$ , or  $(1 + f/ms)$ , since the path does not touch  $L_1$ . The desired transfer function is  $(1/s)(1 + f/ms)(1/\Delta)$ .

Having evaluated all of the required transfer functions, an expression for the output  $X(s)$  can now be written by virtue of Eq. (2.8).

$$\begin{aligned} X(s) = & U(s) \cdot \frac{1/ms^2}{1 + f/ms + k/ms^2} \\ & + v(0) \cdot \frac{1/s^2}{1 + f/ms + k/ms^2} \\ & + x(0) \cdot \frac{(1/s)(1 + f/ms)}{1 + f/ms + k/ms^2} \end{aligned} \quad (2.21)$$

or

$$\begin{aligned} X(s) = & U(s) \cdot \frac{1/m}{s^2 + (f/m)s + k/m} \\ & + v(0) \cdot \frac{1}{s^2 + (f/m)s + k/m} \\ & + x(0) \cdot \frac{s + f/m}{s^2 + (f/m)s + k/m} \end{aligned} \quad (2.22)$$

If the initial conditions  $v(0)$  and  $x(0)$  are zero, Eq. (2.22) reduces to

$$X(s) = U(s) \cdot \frac{1/m}{s^2 + (f/m)s + k/m} \quad (2.23)$$

In the more general case, however, a forcing function  $U(s)$  is specified along with values for the initial conditions  $v(0)$  and  $x(0)$ , and then the inverse Laplace transform of Eq. (2.22) yields the transient response of the system.

Additional drill in the use of Mason's rule can be obtained from the excellent programmed text of signal-flow-graph analysis recently published by Ward and Strum [9]. A discussion of other applications of flow-graph theory and Mason's rule is referred to later chapters.

## REFERENCES

1. J. McCarthy, P. W. Abrahams, D. J. Edwards, T. P. Hart, and M. I. Levin, "LISP 1.5 Programmer's Manual." MIT Press, Cambridge, Massachusetts, 1966.
2. D. P. Friedman, GRASPE graph processing: A LISP extension. Rep. TNN-84. Comput. Center, Univ. of Texas, Austin, 1968.
3. J. Weizenbaum, Symmetric list processor. *Commun. Ass. Comput. Mach.* **6**, 524 (1963).
4. D. J. Farber and R. E. Griswold, SNOBOL, A string manipulation language. *J. Ass. Comput. Mach.* **11**, 21 (1964).
5. "COMIT Programmer's Reference Manual." MIT Press, Cambridge, Massachusetts, 1963.
- 5a. P. M. Lin and G. M. Alderson, SNAP: Symbolic network analysis program. *Proc. Inst. Elec. Eng.* **119**, 160 (1972).
6. S. J. Mason, Feedback theory: Further properties of signal flow graphs. *Proc. IRE* **44**, 920 (1956).
7. D. G. Engelstadt, On Decision Variable Contingency and System Desensitization in Process Design. Ph. D. Dissertation, Univ. of Wisconsin, Madison, Wisconsin, June 1970.
8. D. E. Haskins and C. M. Sliepcevich, *Ind. Eng. Chem. Fundam.* **4**, 245 (1965).
9. J. R. Ward and R. D. Strum, "The Signal Flow Graph in Linear Systems Analysis." Prentice-Hall, Englewood Cliffs, New Jersey, 1968.

This page intentionally left blank

## CHAPTER 3

# SYSTEMATIC ANALYSIS OF LARGE FLOW GRAPHS

Systematic analysis of flow graphs by Mason's rule requires an algorithm for enumerating all the paths and loops in a network. Such an algorithm is desirable in cases where the network is large or where the analysis is to be done with the aid of a computer. Particular attention is paid in this chapter to signal-flow-graph analysis, but the principles apply to the analysis of any network having directed edges.

In the example problems considered in Chapter 2, the signal flow graphs contained few nodes and the detection of the various loops and paths was simple. As more complex problems are considered, their signal-flow-graph representations likewise become more complex. Consider the flow graph shown in Fig. 3.1, which will be discussed further in Chapters 6 and 7. Determining all of the loops and paths in this graph is quite difficult, since it is easy to overlook some in the maze of interconnected branches.



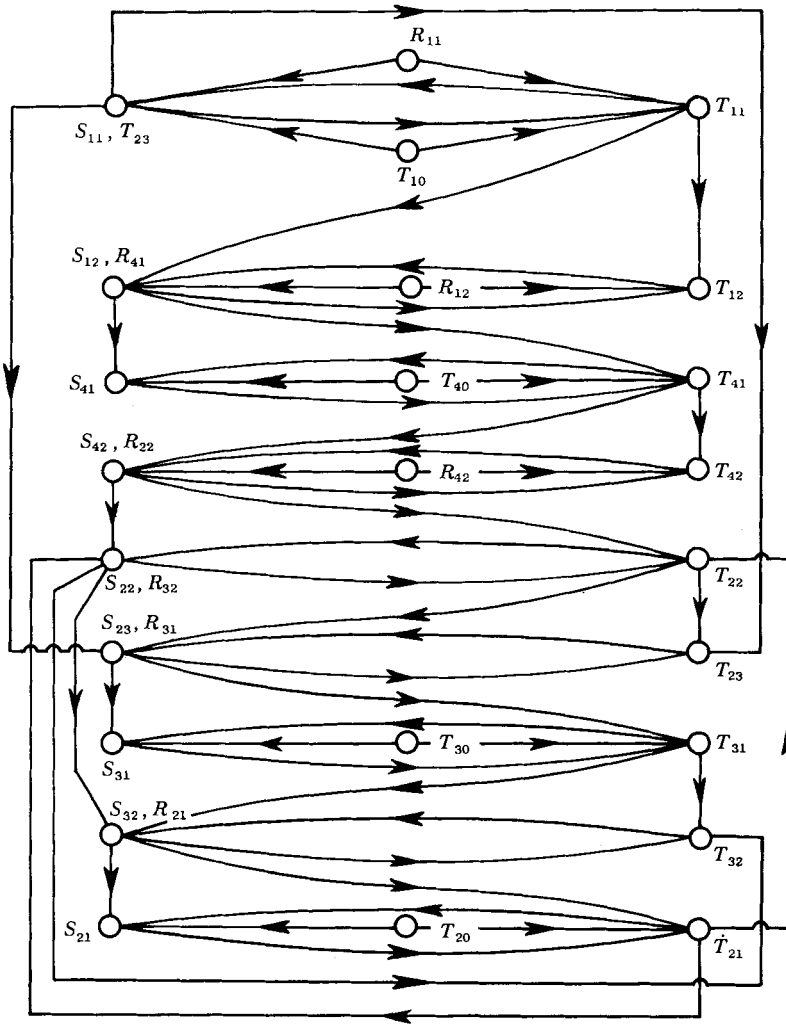


Fig. 3.1. SFG representation of a complex system (gains not shown).

### Determining Specific Paths and Loops

The techniques of linear programming have been applied to specific types of network problems. These methods, which are described in the operations research literature (see, for example, Llewellyn [1]) have found application in solving certain types of transportation problems. Other

algorithms, one of which is described by Moore [2], have been proposed for determining optimal paths through networks. A recent book by Berz-tiss [3] develops some pathfinding algorithms and includes a very comprehensive bibliography.

In this book we focus attention on a general method which leads directly to the explicit definition of every path and loop in a graph and gives the result in an easily usable form. The method is based on a combinational approach to the problem of detecting paths and loops rather than a purely mathematical one. The first point to clarify is that a loop is really just a special type of path, therefore, the distinction between loops and paths can be disregarded and the general problem of finding paths can be considered.

A path is a sequence of branches, but more basically it is a combination of edges. Therefore, if it were possible to determine all of the combinations of all the branches taken 1 through  $N$  at a time, where  $N$  is the total number of branches, and then determine all of the possible permutations of these combinations, all of the possible (and impossible) paths and loops would be included. The bona fida paths and loops would, however, be lost in a multitude of garbage. This garbage would be composed of invalid combinations of branches, and paths in which one or more nodes appear more than once. The trick, then, is to form the combinations and permutations in a manner such that no useless information is generated.

Consider four items:  $a$ ,  $b$ ,  $c$ , and  $d$ . All combinations of these four items can be found in the following manner. First, the items are written as shown below,

List  
\*     $a$     $b$     $c$     $d$   
↓

where  $*$  is as a null item and the first entry in the list of combinations. We proceed by taking each item and appending it to every combination in the list, down to itself, and entering the new combinations at the bottom of the list as they are generated. For the first element, this is a rather trivial operation resulting in the following

List    $a$     $b$     $c$     $d$   
\*  
\* $a$

When a given item is combined with the null item the result is simply that item alone. The null item appears in the list to get the process started and

to insure that the combinations of the  $N$  things taken one at a time will appear in the list of combinations.

Continuing with item  $b$  results in

List    $a$     $b$     $c$     $d$

$*$   
 $\underline{*a}$   
 $\underline{*b}$   
 $\underline{*ab}$

Continuing with item  $c$

List    $a$     $b$     $c$     $d$

$*$   
 $\underline{*a}$   
 $\underline{*b}$   
 $\underline{*ab}$   
 $\left. \begin{array}{l} *c \\ *ac \\ *bc \\ *abc \end{array} \right\} \text{New combinations}$

Note that the new combinations were formed by appending  $c$  to all of the combinations appearing in the list above  $c$ . Finally, after considering item  $d$ , the list is complete and contains all of the possible combinations of these four things taken one through four at a time. The final list is shown below with the null item deleted except as the first element.

List    $a$     $b$     $c$     $d$

$*$   
 $\underline{a}$   
 $\underline{b}$   
 $\underline{ab}$   
 $c$   
 $ac$   
 $bc$   
 $\underline{abc}$   
 $d$   
 $ad$   
 $bd$   
 $abd$   
 $cd$   
 $acd$   
 $bcd$   
 $\underline{abcd}$

The reader should note that thus far, the development says nothing about permutations. This problem will be considered later.

It would be instructive at this point to consider a simple flow graph to see how a procedure similar to that discussed above can be devised to yield all the paths and loops in the system. The graph shown in Fig. 3.2 will serve this purpose. The nodes are numbered, or given addresses, for the sake of simplicity. The nodes will henceforth be referred to by number or address. It is important for purposes of the following discussion that the  $N$  input nodes in a system be arbitrarily numbered 1 through  $N$ .

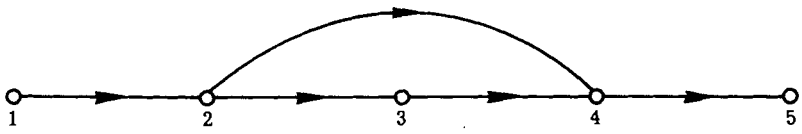


Fig. 3.2. Flow graph containing forward edges only.

An edge which leads to a node with a higher address than its origin will be referred to as a *forward edge*. Edges exhibiting the opposite property will be referred to as *reverse edges*. Figure 3.2 is seen to contain no reverse edges and therefore contains no loops.

The paths in Fig. 3.2 can be systematically generated in the following manner. First, the edges must be ordered in a list such that the edges with the lowest starting addresses appear highest in the list:

- 1 → 2
- 2 → 3
- 2 → 4
- 3 → 4
- 4 → 5

This ordering is necessary so that the edges will be properly permuted as the combinations are formed. This will become more apparent later.

The paths can be generated by forming all possible combinations of the edges in a manner similar to the previous example.

List	1 → 2	2 → 3	2 → 4	3 → 4	4 → 5
*					Path
	<u>1 → 2</u>				1 → 2
	2 → 3				2 → 3
	<u>1 → 2</u>	<u>2 → 3</u>			1 → 2 → 3
	2 → 4				2 → 4

List	1 → 2	2 → 3	2 → 4	3 → 4	4 → 5
1 → 2	2 → 4				1 → 2 → 4
2 → 3	2 → 4				Invalid combination
<u>1 → 2</u>	<u>2 → 3</u>	<u>2 → 4</u>			Invalid combination
3 → 4					3 → 4
1 → 2	3 → 4				Invalid combination
2 → 3	3 → 4				2 → 3 → 4
1 → 2	2 → 3	3 → 4			1 → 2 → 3 → 4
2 → 4	3 → 4				Invalid combination
1 → 2	2 → 4	3 → 4			Invalid combination
2 → 3	2 → 4	3 → 4			Invalid combination
<u>1 → 2</u>	<u>2 → 3</u>	<u>2 → 4</u>	<u>3 → 4</u>		Invalid combination
4 → 5					4 → 5
1 → 2	4 → 5				Invalid combination
2 → 3	4 → 5				Invalid combination
1 → 2	2 → 3	4 → 5			Invalid combination
2 → 4	4 → 5				2 → 4 → 5
1 → 2	2 → 4	4 → 5			1 → 2 → 4 → 5
2 → 3	2 → 4	4 → 5			Invalid combination
1 → 2	2 → 3	2 → 4	4 → 5		Invalid combination
3 → 4	4 → 5				3 → 4 → 5
1 → 2	3 → 4	4 → 5			Invalid combination
2 → 3	3 → 4	4 → 5			2 → 3 → 4 → 5
1 → 2	2 → 3	3 → 4	4 → 5		1 → 2 → 3 → 4 → 5
2 → 4	3 → 4	4 → 5			Invalid combination
1 → 2	2 → 4	3 → 4	4 → 5		Invalid combination
2 → 3	2 → 4	3 → 4	4 → 5		Invalid combination
1 → 2	2 → 3	2 → 4	3 → 4	4 → 5	Invalid combination

In the foregoing list, there are a number of invalid combinations, or combinations which do not form logically valid paths. These can be avoided simply by never appending an edge to a combination in the list unless the ending node of the combination in question matches the starting node of the edge in question.

The reason for ordering the edges should now be apparent. Had the edge  $4 \rightarrow 5$  been first in the list of edges, it would have appeared first in the list of combinations, and no paths ending at node four could have ever been extended onward from there. Therefore, no paths leading to node five would have ever been detected. In summary, the edges must be properly ordered so that the paths are generated starting at the nodes with the lowest addresses and progressing toward the nodes with the highest addresses.

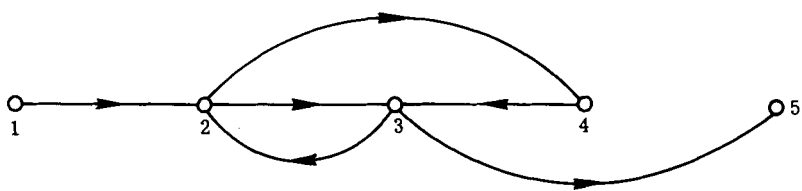


Fig. 3.3. Flow graph containing two loops.

The above example, which corresponds to Fig. 3.2, is simple in that there are no loops. To extend this method to the general case, the flow graph of Fig. 3.3 will be considered. In this case the forward and reverse edges will be ordered separately, the reasons being the same as explained previously.

Forward edges	Reverse edges
$1 \rightarrow 2$	$4 \rightarrow 3$
$2 \rightarrow 3$	$3 \rightarrow 2$
$2 \rightarrow 4$	
$3 \rightarrow 5$	

The procedure for finding the paths is similar to the methods used before except that this time two edge lists will be used. First, the forward paths are found exactly as before using the forward edge list.

List	$1 \rightarrow 2$	$2 \rightarrow 3$	$2 \rightarrow 4$	$3 \rightarrow 5$	(forward)
	$4 \rightarrow 3$	$3 \rightarrow 2$			(reverse)
*					
	<u><math>1 \rightarrow 2</math></u>				
	$2 \rightarrow 3$				
	<u><math>1 \rightarrow 2 \rightarrow 3</math></u>				
	$2 \rightarrow 4$				
	<u><math>1 \rightarrow 2 \rightarrow 4</math></u>				
	$3 \rightarrow 5$				
	$2 \rightarrow 3 \rightarrow 5$				
	<u><math>1 \rightarrow 2 \rightarrow 3 \rightarrow 5</math></u>				

At this point all of the forward paths are known. To find *all* paths, the paths shown above must be allowed to run backward if possible. Therefore, the process is continued using the reverse edges on the next pass. One additional precaution must be taken, however. No path must be generated which falls back on itself unless it forms a loop. An example of this would be the path  $1 \rightarrow 2 \rightarrow 3 \rightarrow 2$ , a path which contains the same node twice and does not define a loop. The path  $2 \rightarrow 3 \rightarrow 2$  would, however, be valid since it forms a loop. Continuing on, using the reverse edge list yields

```

List  1 → 2  2 → 3  2 → 4  3 → 5
      4 → 3  3 → 2
*
1 → 2
2 → 3
1 → 2 → 3
2 → 4
1 → 2 → 4
3 → 5
2 → 3 → 5
1 → 2 → 3 → 5
4 → 3
2 → 4 → 3
1 → 2 → 4 → 3
3 → 2
2 → 3 → 2
4 → 3 → 2
2 → 4 → 3 → 2

```

Thus far, one cycle has been completed in the solution; that is, all paths have been allowed to run forward as far as possible, and these have then been allowed to run backward as far as possible. The solution is not complete, however. The paths which were just generated by allowing forward paths to reverse directions must now be allowed to run forward wherever possible. This is done just as before with only a couple of subtle differences. The list of forward edges will be used and the appending procedure will start at the first path leading backward rather than at the top of the list. Also, nothing will be appended to a loop since this would form an invalid path.

```

List  1 → 2  2 → 3  2 → 4  3 → 5
      4 → 3  3 → 2
*
1 → 2
2 → 3
1 → 2 → 3
2 → 4
1 → 2 → 4
3 → 5
2 → 3 → 5
1 → 2 → 3 → 5
4 → 3
2 → 4 → 3
1 → 2 → 4 → 3
3 → 2
2 → 3 → 2
4 → 3 → 2
2 → 4 → 3 → 2

```

← Starting point for appending

} Paths to be appended to

List	$1 \rightarrow 2$	$2 \rightarrow 3$	$2 \rightarrow 4$	$3 \rightarrow 5$	
	$4 \rightarrow 3$	$3 \rightarrow 2$			
	$3 \rightarrow 2 \rightarrow 3$				} Appending process
	$3 \rightarrow 2 \rightarrow 4$				
	<u><math>4 \rightarrow 3 \rightarrow 2 \rightarrow 4</math></u>				
	<u><math>4 \rightarrow 3 \rightarrow 5</math></u>				
	$2 \rightarrow 4 \rightarrow 3 \rightarrow 5$				
	<u><math>1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 5</math></u>				

The solution is still not complete. The paths just generated in the forward run must now be allowed to reverse directions wherever possible. This procedure must be repeated until a forward or reverse run yields no new paths. Continuing the above problem would reveal another loop in the next reverse run. That loop would be  $3 \rightarrow 2 \rightarrow 4 \rightarrow 3$ . A subsequent forward run would result in no new paths or loops and so the solution is complete.

There is one difficulty in the above algorithm. Notice that two of the loops detected are  $2 \rightarrow 3 \rightarrow 2$  and  $3 \rightarrow 2 \rightarrow 3$ ; two loops which are in fact the same loop! A similar problem is present in the form of loops  $2 \rightarrow 4 \rightarrow 3 \rightarrow 2$ ,  $4 \rightarrow 3 \rightarrow 2 \rightarrow 4$ , and  $3 \rightarrow 2 \rightarrow 4 \rightarrow 3$ ; three loops which are in fact the same loop. This problem is, however, easily solved, but the explanation will be clearer if postponed until later in the development.

### Determining Specific Paths

In certain situations, including the application of Mason's rule, the only paths of interest are those which form loops and those which originate at input nodes. The algorithm previously described yields all paths, not just those which start at input nodes and those which form loops, but also those leading from output nodes to other output nodes. Most of these paths are of no value in the application of Mason's rule. It is possible, however, to alter the proposed algorithm so that the bulk of the unnecessary paths are never generated, and no duplicate loops are generated.

Consider first the problem of finding the paths originating at input nodes. These paths alone can be found simply by allowing no edges to appear alone in the list of combinations except those which originate at an input node. The result can be most easily visualized by considering the previous example problem, Fig. 3.3, in which node 1 is the only input node. Working through the problem just as before, but allowing only input edges to appear alone in the list of combinations yields the following



```

List  1 → 2  2 → 3  2 → 4  3 → 5
      4 → 3  3 → 2

*
1 → 2
1 → 2 → 3
1 → 2 → 4
1 → 2 → 3 → 5
1 → 2 → 4 → 3
1 → 2 → 4 → 3 → 5

```

This procedure has yielded all the paths originating at the input node.

The problem of generating the loops can be handled similarly. It is necessary to first establish a convention to avoid the generation of duplicate loops. This convention states that no loop shall be considered a valid loop unless the first node in the loop has the lowest address of any node appearing in the loop. Therefore, the loop  $2 \rightarrow 3 \rightarrow 2$  is considered valid but the loop  $3 \rightarrow 2 \rightarrow 3$  is not.

The procedure for finding the loops with the bulk of the unnecessary paths deleted is based on the following two modifications to the path finding algorithm.

First, only those forward edges whose starting nodes are the same as the terminal node of some reverse edge are allowed to appear alone in the list of combinations. For example, the edge  $3 \rightarrow 6$  can appear alone in a list of combinations only if an edge such as  $4 \rightarrow 3$  exists somewhere in the network under consideration. The reason for this is to avoid generating paths which have no hope of ever closing to form loops.

The second modification requires that no path be generated which ends with a node having an address lower than that of the path's starting node. This prevents the formation of duplicate loops by insuring that when a loop is formed, the starting node will have the lowest address of any node in the loop.

Paths leading from input nodes to output nodes and paths forming loops can be generated independently via the same algorithm by governing the type of edges which are allowed to appear alone in the list. The second modification mentioned above requires, however, that the  $N$  input nodes be assigned the addresses 1 through  $N$ . If this were not the case, some of the paths leading from inputs to outputs would exhibit the forbidden property of containing nodes having lower addresses than their starting address.

Applying the modified loop finding procedure to the signal flow graph shown in Fig. 3.3 reveals the loops.

List	1 → 2	2 → 3	2 → 4	3 → 5
	4 → 3	3 → 2		
*				
<u>2 → 3</u>				Necessarily generated
<u>2 → 4</u>				Necessarily generated
<u>3 → 5</u>				Unnecessarily generated
<u>2 → 3 → 5</u>				Unnecessarily generated
<u>2 → 4 → 3</u>				Necessarily generated
<u>2 → 3 → 2</u>				Loop
<u>2 → 4 → 3 → 2</u>				Loop
<u>2 → 4 → 3 → 5</u>				Unnecessarily generated

It is true that a number of paths are generated unnecessarily by this procedure. This is, unfortunately, unavoidable, but the number of such paths is still much smaller than the number generated by the unmodified procedure. The saving grace is that the unnecessary paths can be quickly eliminated, as will be demonstrated shortly.

The manner in which the above procedures lend themselves to the independent determination of loops, and paths starting at a given input node, make these procedures particularly valuable in applying Mason's rule. The loops can be easily found, the unnecessary paths eliminated, and the necessary information stored for future reference in computing the determinant  $\Delta$ . The paths starting at any specified input node can be generated, used, and then eliminated to make space for paths originating at the next input node to be considered. This scheme makes it possible to handle graphs much larger than those which can be handled when all paths and loops must be stored simultaneously.

**List Processing Approach to Determining Paths**

The list generated for Fig. 3.3 can be written in matrix form as

$$\begin{bmatrix} 2 & 3 & & & \\ 2 & 4 & & & \\ 3 & 5 & & & \\ 2 & 3 & 5 & & \\ 2 & 4 & 3 & & \\ 2 & 3 & 2 & & \\ 2 & 4 & 3 & 2 & \\ 2 & 4 & 3 & 5 & \end{bmatrix}$$

This method of storing the paths is wasteful of storage since a number of

blank locations are present. In larger systems where paths are longer this situation is aggravated. The problem can, however, be satisfactorily resolved. In the following table the paths are written on the left as before—on the right is an explanation of what each path really is.

Row	Path	Composition
1	$2 \rightarrow 3$	Edge $2 \rightarrow 3$
2	$2 \rightarrow 4$	Edge $2 \rightarrow 4$
3	$3 \rightarrow 5$	Edge $3 \rightarrow 5$
4	$2 \rightarrow 3 \rightarrow 5$	Row 1 with 5 appended
5	$2 \rightarrow 4 \rightarrow 3$	Row 2 with 3 appended
6	$2 \rightarrow 3 \rightarrow 2$	Row 1 with 2 appended
7	$2 \rightarrow 4 \rightarrow 3 \rightarrow 2$	Row 5 with 2 appended
8	$2 \rightarrow 4 \rightarrow 3 \rightarrow 5$	Row 5 with 5 appended

The last column of the above table indicates that a path can be represented as a path appearing in a previous row with one node appended. The only exception is the paths which are composed of single edges. When the information is stored in this manner any path can be traced easily. Consider the path shown in row 8 of the table above.

The last node appended is found on row 8 and is node 5. This row points back to row 5 where node 3 is encountered. At this point, nodes 5 and 3 have been detected. Row 5 points back to row 2 which contains the edge  $2 \rightarrow 4$ . Row 2 points to no other rows so the trace is complete. The trace was made in reverse order, however, so the path represented by row 8 is really  $2 \rightarrow 4 \rightarrow 3 \rightarrow 5$ . It is now evident that all paths can be contained in a small matrix with no wasted space, as shown below.

Row	Row Pointer	Appended Node
1	1	3
2	2	4
3	3	5
4	1	5
5	2	3
6	1	2
7	5	2
8	5	5

The entries above which correspond to edges simply point to themselves, and the node referenced in the second column is the terminal node of the edge. The node corresponding to the start of the edges is, however, not listed. To include this information another column is required. In this column the

origin of each path is recorded. This column is easily filled in as the list is generated and it is helpful not only in indicating the origin of the edge entries, but also in telling where the various paths start and which paths are loops, without having to completely trace them back to their origins. Adding the origin column to the previous list yields

Row	Origin	End	Row Pointer
1	2	3	1
2	2	4	2
3	3	5	3
4	2	5	1
5	2	3	2
6	2	2	1
7	2	2	5
8	2	5	5

When paths are stored in the list structure shown above, each path can be represented by three numbers, or one row in the matrix. This leads to tremendous savings in core storage when the graph analysis is done by computer. This is particularly true if half-word integers are used to store the numbers.

The generation of the paths using the list structure is exactly analogous to the procedures already explained and demonstrated by example. The addresses of the first and last nodes of each path are readily available, and it is these addresses that are used in the tests made while generating the paths. The procedure is actually made simpler in one respect. It is necessary to record only the row number of a path which is being extended rather than copying the entire path.

### Evaluation of Gains

Loop gains or path gains can be computed very readily as the various paths and loops are generated. Each branch gain is of the form  $as^n$ , where  $a$  is some constant, and  $n$  is the power of the Laplace variable  $s$ . Since the path gains and loop gains are products of a series of branch gains, they too will be of the form  $as^n$ . Any such gain can then be stored as a constant  $a$ , and a power of  $s$  to  $n$ . The gains themselves are computed as the paths and loops are generated by simply performing one multiplication and one addition each time an edge is appended to another edge or path. The power of  $s$  corresponding to each entry in the list structure is conveniently stored

in an additional column of the list structure matrix, and the constants  $a$  can be stored in a parallel vector.

In the evaluation of the system determinants, which will be discussed in a later section, it will be necessary to form the product of a number of loop gains. These multiplications can be performed, and the results stored, as described above.

### **Removing Unnecessary Paths from the List Structure**

In a previous section, the problem of eliminating the unnecessary paths generated in the loop finding process was deferred. This problem can now be considered and resolved by applying the list structure concept.

In forming the list structure, the column used to store the  $n$ 's, or the powers of  $s$ , can double as a status vector. The important  $n$ 's, those which correspond to the entries representing closed loops, can be temporarily stored in the "end" column since the origin and the end of a loop are the same.

In eliminating the unnecessary paths, the matrix is simply scanned from top to bottom. The status of each row is set to 0, indicating an unnecessary path, until a loop is encountered. Loops are indicated by the fact that the starting and ending nodes are the same. When a loop is detected it is traced backward and the status of each row encountered is set to 1, indicating a necessary path. This process is continued until the bottom of the list is reached. The necessary rows will all have a status of 1 at this point.

The rows having a status of 0 can now be considered vacant and the valid rows can be moved upward in the list. Care must be taken, however, to update the row pointers of the valid rows during this procedure.

The list is again scanned from top to bottom. When a path with a status of 1 is encountered its origin and end are copied into the vacant row nearest the top of the list, the row pointer of the vacant row is set equal to the number in the status column of the row pointed to by the row being copied, and the number in the status column of the row being copied is set equal to the number of the vacant row into which it is being copied. This procedure ensures that every row points to the new location of the row it pointed to before reshuffling. If a row of status 1 is encountered and there are no vacant rows above it, its fourth element is simply set equal to the row number and the scan continues.

There are other ways to perform this operation which are conceptually simpler, but they are not nearly as efficient.

The garbage elimination problem becomes important only when storage must be conserved. In many cases the unnecessary paths can be left in the list with impunity.

Evaluating Determinants in Mason's Formula

In Chapter 2 the determinant  $\Delta$  appearing in Mason's rule was seen to be made up of combinations of loops taken 1 through  $N$  at a time. Also, procedures were presented for finding combinations of items taken 1 through  $N$  at a time, and for finding all of the loops in a network. Therefore, all of the information needed to compute  $\Delta$  for a given network is at hand. Furthermore, the loop combinations can be generated using the list structure concept.

The first problem is that of determining when two loops touch. First, a vector of 0's and 1's is established. The locations containing 1's indicate the nodes present in the first loop. The vector for the loop  $2 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow 2$  is

1	2	3	4	5	6	7	8	9	...	$N$
0	1	1	1	0	1	0	0	0	...	0

The second loop is simply traced and the nodes encountered are compared with the contents of the vector. If a node is encountered such that the corresponding location in the vector contains a 1, then the two loops touch. Consider the loop  $5 \rightarrow 7 \rightarrow 9 \rightarrow 6 \rightarrow 5$  in relation to the vector shown above. Locations 5, 7, and 9 contain 0's, but location 6 contains a 1, so the two loops touch. This operation is rapid, for once the nodes corresponding to a loop are loaded into the vector, comparison can be made with many other loops without repeating this procedure. Also, loops most often will touch, and this will be detected, as often as not, before the loop under consideration is traced halfway along its course.

Consider a system containing four loops  $L_1$ ,  $L_2$ ,  $L_3$ , and  $L_4$ . The problem is to generate all possible combinations of loops which do not contain loops that touch. A list structure can be used as before.

Loop	Pointer	Status
↓	↓	↓

In a given row the column labeled *loop* contains the number of the loop which was appended to some other loop combination to form the combina-

tion represented by that row. The pointer column in the row indicates the row to which the loop was appended. Combinations which are composed of single loops point to themselves. The column labeled *status* is used to indicate whether or not a given loop touches any loop in the combination represented by that row.

The procedure begins, as before, by placing the information regarding the first loop on the first row.

Combination	Row	Loop	Pointer	Status
$L_1$	1	1	1	—

The status column is left blank for the time being; it will be used later. The information regarding the second loop is entered on the second row.

Combination	Row	Loop	Pointer	Status
$L_1$	1	1	1	—
$L_2$	2	2	2	—

The second loop is now combined with everything in the list above it and the combinations are appended to the bottom of the list so long as no touching loops are included in the combination. The first step then is to test  $L_2$  against  $L_1$ , to see if they touch. Suppose that they do not touch. The status of row 1, that is  $L_1$ , can be set equal to 1, indicating that  $L_2$  does not touch  $L_1$ . The third row of the list structure would then represent the combination  $L_1L_2$ .

Combination	Row	Loop	Pointer	Status
$L_1$	1	1	1	1
$L_2$	2	2	2	—
$L_1L_2$	3	2	1	—

$L_2$  has been combined with  $L_1$ , the only combination in the list above itself, so the process continues with  $L_3$ .

$L_3$  is entered in the list.

Combination	Row	Loop	Pointer	Status
$L_1$	1	1	1	1
$L_2$	2	2	2	—
$L_1L_2$	3	2	1	—
$L_3$	4	3	4	—

$L_3$  must now be combined with everything appearing above it in the list. First,  $L_3$  is compared with  $L_1$  to see if they touch. If they touch, the status of row one must be set equal to 0 to indicate that  $L_3$  touches  $L_1$ . The entry

$L_1L_3$  is not made on the bottom of the list since these loops touch. Next,  $L_3$  is compared with  $L_2$ . Suppose they are found not to touch. The status of row 2 is set equal to 1 and the combination  $L_2L_3$  is added to the bottom of the list structure, yielding

Combination	Row	Loop	Pointer	Status
$L_1$	1	1	1	0
$L_2$	2	2	2	1
$L_1L_2$	3	2	1	—
$L_3$	4	3	4	—
$L_2L_3$	5	3	2	—

Next,  $L_3$  must be combined with the next row, which represents the combination  $L_1L_2$ . It was just discovered that  $L_3$  did not touch  $L_2$ , but a check must be made to assure that no other loop in the combination is touched by  $L_3$ . This is done by tracing one step back up the list structure to see if the row pointed to has a 0 status with respect to  $L_3$ . The combination  $L_1L_2$  represented by row 3 points back to row 1 which has a status of 0. That status was set when  $L_3$  was seen to touch  $L_1$ . Therefore, the combination  $L_1L_2L_3$  is invalid and is not added to the bottom of the list structure. The status of row 3 must be set to 0.  $L_3$  has now been combined with everything above it in the list so  $L_4$  is considered next. Adding  $L_4$  to the bottom of the list structure yields

Combination	Row	Loop	Pointer	Status
$L_1$	1	1	1	0
$L_2$	2	2	2	1
$L_1L_2$	3	2	1	0
$L_3$	4	3	4	—
$L_2L_3$	5	3	2	—
$L_4$	6	4	6	—

$L_4$  must now be appended to the combinations above it. If  $L_4$  is not seen to touch  $L_1$ , the status of row 1 is set equal to 1 and a new row is added to the list structure.

Combination	Row	Loop	Pointer	Status
$L_1$	1	1	1	1
$L_2$	2	2	2	1
$L_1L_2$	3	2	1	0
$L_3$	4	3	4	—
$L_2L_3$	5	3	2	—
$L_4$	6	3	6	—
$L_1L_4$	7	4	1	—



Next  $L_4$  is compared to  $L_2$ . Suppose they touch. The status of rows 2 and 3 can immediately be set to 0 since both of these combinations end with  $L_2$ . The same would be true of rows 4 and 5 if  $L_4$  was found to touch  $L_3$ . No new entries would then be made to the bottom of the list.

With this technique, all of the terms required to compute the  $\Delta$  for a given signal flow graph are determined rapidly and efficiently by sequentially picking the loops out of the list structure containing the loops, and combining them. The numerical or functional value of each term is determined as the terms are generated, in a manner similar to the technique described for computing the path and loop gains. The only difference is that the value corresponding to a single loop is taken as the negative of the actual loop gain, as required by Eq. (2.11). The signs associated with subsequent terms are obtained as the negative of the sign of the term to which they point.

The expression for the system determinant is obtained by adding all coefficients of like powers of  $s$  in the delta list structure to yield the coefficients of a polynomial in  $s$ . In the case of static systems, all of the powers of  $s$  will be zero and the value of the system determinant will be obtained as a single number.

### Computing the Determinants Corresponding to Specific Paths as Required by Mason's Rule

When applying Mason's rule, it is necessary to eliminate the terms from  $\Delta$  which contain loops touched by a specific path to obtain the  $\Delta_k$  corresponding to that path. This can be done once the terms of  $\Delta$  are established in the list structure. The procedure simply involves treating the path just as the loops were treated in generating  $\Delta$ . The path is compared with each individual loop appearing in the delta list structure, and the status of each term in  $\Delta$  with respect to the path is obtained. The  $\Delta$  for the path is then computed by summing the terms in the delta list structure which have a status of 1. The only difference between this procedure and the procedure for generating the terms in  $\Delta$  is that nothing is ever added to the list structure.

### Computing Transmittances

The transmittance or transfer function relating a given input to a given output is obtained from Mason's rule, Eq. (2.10). The summation  $\sum P_k \Delta_k$  in the numerator of Eq. (2.10) will yield a polynomial in  $s$  if the system is a

dynamic one, otherwise, it will yield a pure number. In the case of a polynomial, the coefficients of the various powers of  $s$  are generated by considering each path from the input to the output. The terms of  $\Delta$  which constitute  $\Delta_k$  for the  $k$ th such path are obtained as described in the previous section. Multiplying each term of  $\Delta_k$  by  $P_k$  and summing like powers of  $s$ , for all  $k$ , will yield the coefficients of the desired polynomial. The transfer function is then obtained as a ratio of two polynomials:

$$T(s) = (\sum P_k \Delta_k) / \Delta = N(s) / D(s) \quad (3.1)$$

### Implementing the Algorithms

At this point algorithms are available for computing transmittances, determining loops and paths, and the systematic application of Mason's rule. These algorithms have been programmed and implemented on a digital computer. The ability to compute transmittances individually is helpful when computing the values of the outputs individually and when performing sensitivity studies. A schematic representation of the mechanics involved in the program is presented in Fig. 3.4. Even though this program is basically concerned with the application of Mason's rule, the various subroutines which constitute the program can be used in other applications.

The most important programming consideration involves the data structure. Figure 3.5 illustrates how an array  $P$  and a parallel vector  $G$  are used to store the various list structures and gains. The  $P$  array and the  $G$  vector are partitioned into several smaller segments for a very important reason. Before execution of a given problem it is impossible to determine exactly how long the various list structures will be, and for this reason a means of obtaining dynamic storage allocation in FORTRAN is needed to insure efficient use of available storage. The concept of one large matrix which can be partitioned during execution provides this capability.

The programs in Fig. 3.4 are written in  $G$  level FORTRAN and consist of seven basic subroutines. The nature of the MAIN program depends on the application since it can call one or more of these subroutines. Appendix I contains a description of each subroutine. Schematic diagrams, and a listing of the corresponding FORTRAN program are available from the authors. In addition to the basic subroutines, there are several others which are useful for specific applications. These will be discussed in later chapters.

The program is composed of about 500 cards. Briefer versions are certainly conceivable since this program contains comment cards, extensive

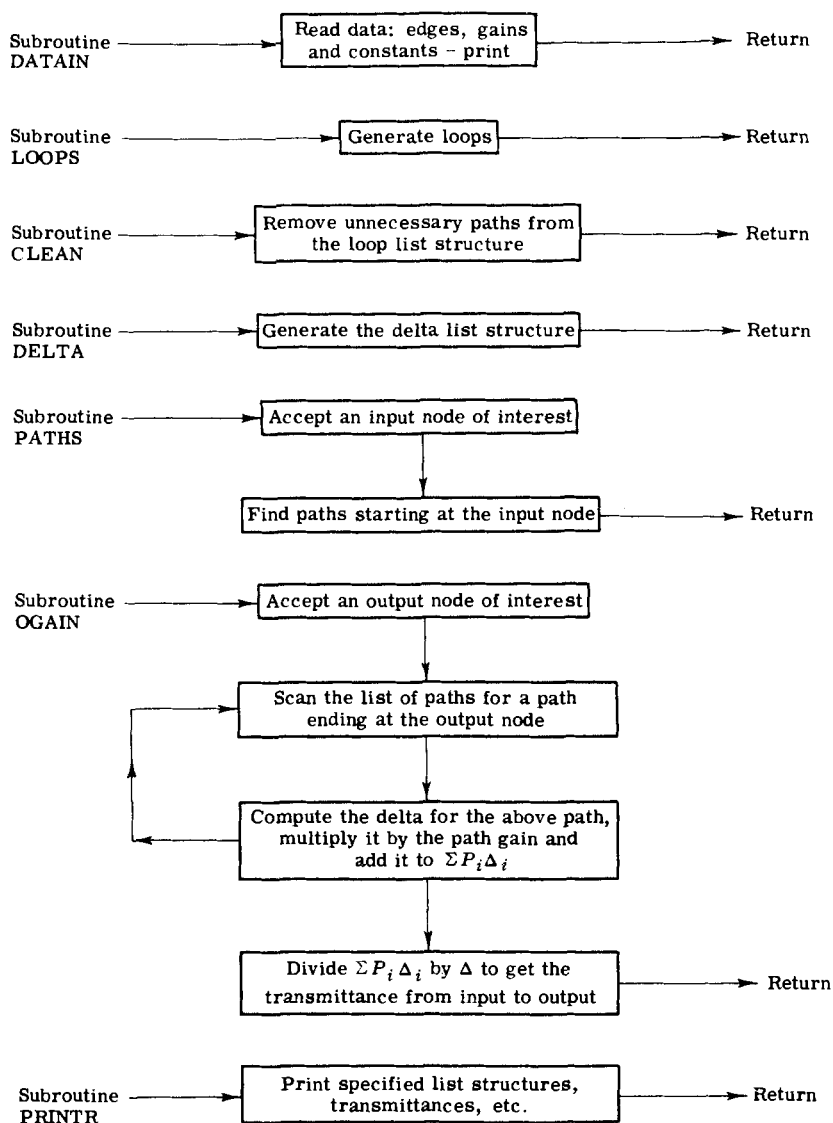


Fig. 3.4. Schematic diagram for a program to compute specified transmittances.

Partition	Row	<i>P</i> Array				<i>G</i> Vector
Edges which constitute the graph	1					
	.					
	.					
	.					
	FREV	Origin	End	Pointer	Power	Branch gain
	.					
List structure containing the loops	.					
	.					
	NEDGS	Origin	End	Pointer	Power	Path or loop gain
	.					
	.					
	LEND					
Delta list structure	DSTART					
	.					
	.					
	.	Loop	Pointer	Power	Status	Loop or loop combination gain
	.					
	DEND					
List structure containing the paths starting at a specified input node	DEND 1					
	.					
	.	Origin	End	Pointer	Power	Path gain
	.					
	.					
	LEND					

Fig. 3.5. Partitioning and contents of the *P* array and the *G* vector.

printing capabilities, and diagnostic messages. The core requirement varies depending on the length specified for the *P* array and *G* vector. The 32K word memory of an IBM 360/44 is sufficient to contain the resident operating system, the basic subroutines, and a *P* array and *G* vector containing in excess of 6000 rows—without overlay.

## REFERENCES

1. R. W. Llewellyn, "Linear Programming," p. 335. Holt, New York, 1964.
2. E. F. Moore, The shortest path through a maze. *Proc. Int. Symp. Theory of Switching*, Cambridge, Massachusetts, April 2-5, 1957, p. 285. Harvard Univ. Press, Cambridge Massachusetts, 1959.
3. A. T. Berziss, "Data Structures, Theory and Practice." Academic Press, New York, 1971.

This page intentionally left blank

## CHAPTER 4

# FREQUENCY RESPONSE ANALYSIS

For linear dynamic systems the transfer function relating a given input to a given output is generally obtained as a ratio of two polynomials in  $s$ , the Laplace variable. Ordinarily, the transfer function is multiplied by the Laplace transform of the input  $I(s)$  and the resultant output function  $X(s)$  is inverted to yield the contribution of the input to the output in the time domain  $x(t)$ . It is, however, often unnecessary to perform the inversion to glean useful information about a system. In this chapter a brief review of transfer function analysis in the  $s$  domain is presented. It is suggested that a reader conversant with this topic proceed directly to Chapter 5.

### Final Value Theorem

If  $X(s)$  is the Laplace transform of  $x(t)$ , then

$$\lim_{t \rightarrow \infty} \{x(t)\} = \lim_{s \rightarrow 0} \{sX(s)\} \quad (4.1)$$

provided that  $sX(s)$  does not become infinite for any value of  $s$  satisfying

$\text{Re}(s) \geq 0$ , where  $\text{Re}$  denotes the real part of the complex function. If this is not true,  $f(t)$  does not approach a limit as  $t$  becomes infinite.

This theorem provides a simple means of determining the effect of a unit step change in some input on the steady state values of the various outputs. If

$$T(s) = X(s)/I(s) \quad (4.2)$$

then

$$X(s) = I(s) \cdot T(s) \quad (4.3)$$

For a step input of unity,

$$I(s) = 1/s \quad (4.4)$$

therefore, from Eq. (4.3),

$$X(s) = (1/s)T(s) \quad (4.5)$$

from which it is evident that

$$sX(s) = T(s) \quad (4.6)$$

and

$$\lim_{s \rightarrow 0} \{sX(s)\} = \lim_{s \rightarrow 0} \{T(s)\} = \lim_{t \rightarrow \infty} \{x(t)\} \quad (4.7)$$

To recapitulate, the contribution of a step input of unity to the steady state value of an output is obtained as the limit of the corresponding transfer function as  $s$  approaches zero. This value is, by definition, the steady state gain of the transfer function  $T(s)$ . If the transfer function of interest is a ratio of two polynomials in  $s$ , the desired limit is simply equal to the ratio of the constants appearing in the numerator and denominator polynomials.

### Initial Value Theorem

If  $X(s)$  is the Laplace transform of  $x(t)$ , then

$$\lim_{t \rightarrow 0} \{x(t)\} = \lim_{s \rightarrow \infty} \{sX(s)\} \quad (4.8)$$

or, by Eq. (4.6)

$$\lim_{t \rightarrow 0} \{x(t)\} = \lim_{s \rightarrow \infty} \{T(s)\} \quad (4.9)$$

Equation (4.9) is useful in determining the initial gain of the transfer func-

tion  $T(s)$  at  $t = 0$ . If the denominator of  $T(s)$  is a polynomial of degree  $n$ , and the numerator is a polynomial of degree  $m$ , where  $m = n$ , the desired limit is evaluated as the ratio of the two  $n$ th-order coefficients. If  $m < n$ , the limit has a value of zero. The case where  $m > n$  will not ordinarily arise.

### Substitution Rule

Consider a first-order system which has the transfer function

$$T(s) = X(s)/I(s) = 1/(\tau s + 1) \quad (4.10)$$

where  $\tau$  is a time constant. If this system is subjected to a sinusoidal input having an amplitude  $A$  and an angular frequency  $\omega$ ,

$$I(t) = A \sin \omega t \quad (4.11)$$

the steady-state response of the system, as obtained by conventional methods, will be

$$x(t) = \frac{A}{[(\tau\omega)^2 + 1]^{1/2}} \cdot \sin(\omega t + \phi) \quad (4.12)$$

where

$$\phi = \tan^{-1}(-\omega\tau) \quad (4.13)$$

The amplitude ratio  $AR$  which is the ratio of the output amplitude to the input amplitude is

$$AR = 1/[(\omega\tau)^2 + 1]^{1/2} \quad (4.14)$$

and the *phase angle* between the input and the output is given by Eq. (4.13).

If  $j\omega$  is substituted for  $s$  in Eq. (4.10), we obtain:

$$|T(j\omega)| = 1/[(\omega\tau)^2 + 1]^{1/2} \quad (4.15)$$

$$\angle T(j\omega) = \tan^{-1}(-\omega\tau) \quad (4.16)$$

This simple substitution results in a complex number which has a magnitude and argument identical to the amplitude ratio and phase angle shown in Eqs. (4.14) and (4.13), respectively. It is possible, then, by means of this substitution, to obtain the amplitude ratio and the phase angle corresponding to a sinusoidal input without having to resort to a rigorous analysis. This result is very useful, and the technique is applicable to higher order systems.



**Bode Diagrams**

A given transfer function  $T(s)$  can be converted to the  $T(j\omega)$  form by substituting  $j\omega$  for  $s$  everywhere it appears in  $T(s)$ . If the magnitude and argument of  $T(j\omega)$  are evaluated for a series of values of  $\omega$ , the information required to generate plots of amplitude ratio and phase angle versus frequency is obtained. A plot of amplitude ratio versus frequency, along with a plot of phase angle versus frequency, constitute the *Bode diagrams*. These

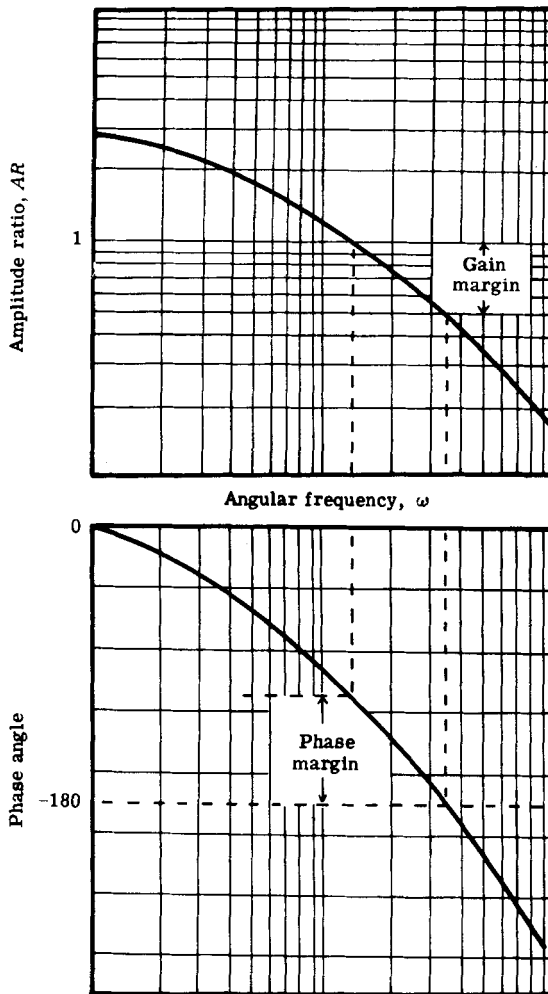


Fig. 4.1. A typical open-loop Bode diagram.

diagrams find wide application in control systems analysis. Bode diagrams generally fall into two categories: *open loop* and *closed loop*.

Open-loop Bode diagrams, Fig. 4.1, are generated by considering the transfer function relating the input of interest to the output of interest. This transfer function is obtained by analyzing the system with the feedback to the controller eliminated. Open-loop diagrams are very useful in the application of *Bode's stability criterion* which is stated as follows: A control system is unstable if the open-loop Bode diagram exhibits an amplitude ratio

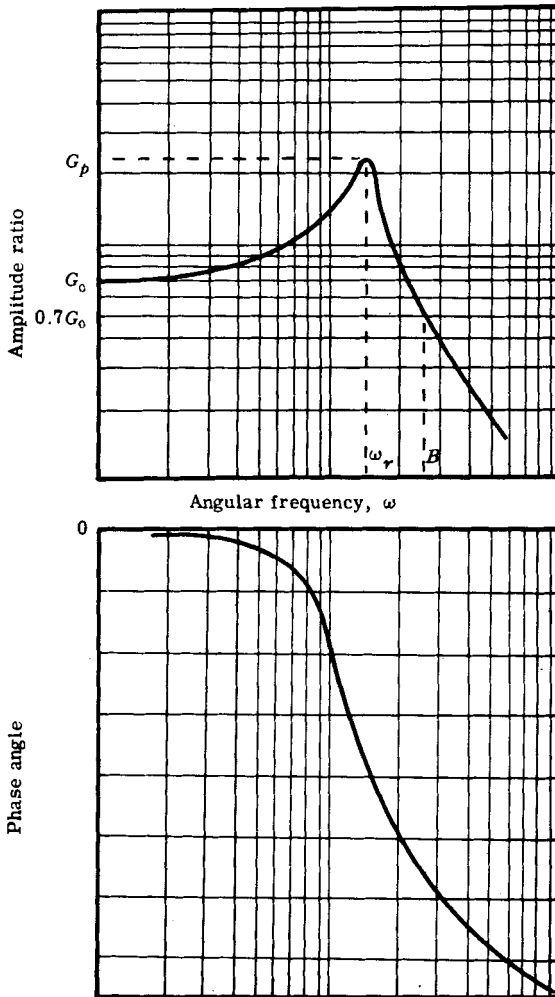


Fig. 4.2. A typical closed-loop Bode diagram.

greater than unity at the frequency for which the phase angle is  $-180^\circ$ . This rule is based on the argument that, where the specified conditions occur, the error seen by the controller will be continually amplified, which results in an unstable situation. The Bode stability criterion is not a general rule and should only be applied in cases where the phase angle is a monotonically decreasing function of frequency.

Closed-loop Bode diagrams are generated from transfer functions obtained by analyzing the system with the feedback to the controller connected. These diagrams provide considerable qualitative information about the step response of a system. Figure 4.2 pictures a typical closed-loop Bode diagram.  $G_0$  is the steady state gain of the transfer function, and  $G_p$  is the height of the resonant peak which appears at  $\omega_r$ , the resonant frequency. The bandwidth  $B$  is the frequency at which the amplitude ratio  $AR$  equals  $0.7G_0$ .

It is possible to make some generalization about the relationship between closed-loop Bode diagrams and their corresponding step responses. Systems which exhibit a high resonant peak will, in general, exhibit a less damped, more oscillatory response than if the peak were lower and flatter. The oscillations of systems having a high value of  $\omega_r$  will, in general, have a shorter period of oscillation than they would if  $\omega_r$  occurred at a lower frequency. Systems exhibiting high values of the bandwidth  $B$  tend to respond more rapidly than systems with lower values of  $B$ . While these generalizations are often useful in designing control systems, they must be applied with caution because, under certain circumstances, they can lead to erroneous conclusions.

### A Typical Application

The use of frequency response techniques to study distillation column dynamics was recently discussed in the literature [1]. The various transfer functions  $T(j\omega)$  yield closed-loop Bode diagrams which are used to provide insight into the characteristics of the column. Results for  $\omega = 0$  give the steady state gains of the transfer functions which correspond to step changes in the independent variables. These steady-state gains reveal the significance of disturbances in variables such as feed rate, enthalpy, heat losses, reboiler duty and cooling water temperature, and indicate how the steady-state profiles of temperature, composition, and flow rates respond.

In the analysis of distillation dynamics, the Bode diagrams are generated over a range of frequencies which are appropriate for the disturbance in

question. High frequencies are used when looking at hydraulic disturbances because this type of disturbance propagates itself through the column very rapidly as compared to composition disturbances, which must be investigated at lower frequencies. These diagrams then, are useful in predicting the nature of the transient response, selecting controller settings to achieve the desired operating characteristics, and in improving the mathematical model of the process.

### Other Stability Criteria

When Mason's rule is used to obtain closed-loop transfer functions, the same function,  $\Delta$ , appears in the denominator of each transfer function. This function, when equated to zero, is generally referred to as the *characteristic equation* of the system. The nature of this equation dictates the transient response of the system since the real parts of its roots appear in exponential exponents of the expression for the time domain transient response. If the roots of the characteristic equation all have negative real parts, a transient response which decays with time results and the system is stable. If any root of the characteristic equation has a positive real part, a transient response which grows exponentially with time results and the system is unstable. It is possible, therefore, to determine stability by looking at the roots of the characteristic equation.

It is possible to see the effect of parameter variations on a system's stability by plotting a *root locus diagram*. A root locus diagram is a plot, in the complex plane, of the roots of the characteristic equation for a series of values for the parameter of interest. Root locus diagrams are used to show the effect of parameter variations and to predict *critical values*; that is, values of the parameter beyond which the real part of one or more roots becomes positive.

Another stability criterion which is widely used is the *Nyquist stability criterion*. This criterion is very similar to the Bode stability criterion, but it is more generally applicable. In applying this technique, a *Nyquist plot* is prepared. Nyquist plots are plots, in the complex plane, of the real and imaginary parts of the transfer function of interest,  $T(j\omega)$ , for a series of values of  $\omega$ . If the Nyquist plot of a function encircles the point  $(-1, 0)$ , the corresponding transient response will be unstable.

The utility of the techniques discussed in this chapter are enhanced by the ability to compute the required transfer functions in an efficient manner. The signal flow graph, Mason's rule, and the methods of imple-

mentation presented in Chapter 4 provide this capability, as will be shown in Chapter 6.

#### REFERENCE

1. R. E. Bollinger, Predicting fractionator dynamics by using a frequency domain solution technique. *AIChE J.* **16**, 673 (1970).

## CHAPTER 5

# SENSITIVITY ANALYSIS

The objective of a sensitivity analysis is to determine the effect of variations in the system parameters on the outputs, or on some performance criteria. This chapter deals with the application of signal flow graphs and their associated transfer functions in computing sensitivities. Important sensitivity functions are derived and techniques of implementing them are discussed.

### Sensitivity Analysis in General

The mathematical model for a dynamic system can be expressed as a set of  $N$  functions involving time  $t$ , a vector of  $N$  state variables  $\mathbf{X}$ , its time derivatives  $\dot{\mathbf{X}}$ ,  $\ddot{\mathbf{X}}$ , etc., and a particular parameter of interest  $p$ ; the parameters other than  $p$  manifesting themselves in the functions as constants which are based on the nominal values of the parameters. Each function is then of the form

$$F_k(p, t, \mathbf{X}, \dot{\mathbf{X}}, \ddot{\mathbf{X}}, \dots) = 0 \quad (5.1)$$

Taking the derivative of Eq. (5.1) with respect to  $p$  yields

$$\frac{\partial F_k}{\partial p} + \frac{\partial F_k}{\partial \mathbf{X}} \frac{\partial \mathbf{X}}{\partial p} + \frac{\partial F_k}{\partial \dot{\mathbf{X}}} \frac{\partial \dot{\mathbf{X}}}{\partial p} + \frac{\partial F_k}{\partial \ddot{\mathbf{X}}} \frac{\partial \ddot{\mathbf{X}}}{\partial p} + \dots = 0 \quad (5.2)$$

defining the vector of desired sensitivities as  $\mathbf{U}$ ,

$$\frac{\partial \mathbf{X}}{\partial p} = \mathbf{U} \quad (5.3)$$

$$\frac{\partial \dot{\mathbf{X}}}{\partial p} = \frac{d}{dt} \frac{\partial \mathbf{X}}{\partial p} = \dot{\mathbf{U}} \quad (5.4)$$

$$\frac{\partial \ddot{\mathbf{X}}}{\partial p} = \frac{d^2}{dt^2} \frac{\partial \mathbf{X}}{\partial p} = \ddot{\mathbf{U}}, \text{ etc.} \quad (5.5)$$

Substituting in Eq. (5.2) yields

$$\frac{\partial F_k}{\partial \mathbf{X}} \mathbf{U} + \frac{\partial F_k}{\partial \dot{\mathbf{X}}} \dot{\mathbf{U}} + \frac{\partial F_k}{\partial \ddot{\mathbf{X}}} \ddot{\mathbf{U}} + \dots = - \frac{\partial F_k}{\partial p} \quad (5.6)$$

Equation (5.6) represents a system of  $N$  equations which can be solved simultaneously to yield  $\mathbf{U}$ ,  $\dot{\mathbf{U}}$ ,  $\ddot{\mathbf{U}}$ , etc. The derivatives of  $F_k$  which appear in Eq. (5.6) can be evaluated directly from Eq. (5.1); the parameter  $p$  being evaluated at its nominal value. The initial condition for the sensitivities is  $\mathbf{U} = \dot{\mathbf{U}} = \ddot{\mathbf{U}} = 0$ . A repetitive solution of Eq. (5.6), changing only the right-hand sides each time enables one to obtain the sensitivity vectors associated with any parameter, provided the solution of Eq. (5.1) is analytically dependent on the parameter.

A simple example of sensitivity analysis devised by Engelstadt [1] illustrates the application of these equations. For the two tanks shown in Fig.

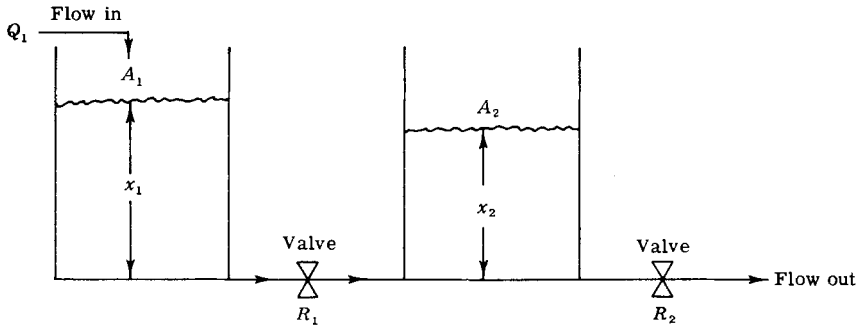


Fig. 5.1. A two-tank system.

5.1,  $x_1$  and  $x_2$  are the liquid heights.  $Q_1$  is the inlet flow to tank one,  $R_1$  and  $R_2$  represent valve resistances, and  $A_1$  and  $A_2$  are tank areas. We wish to obtain the sensitivity  $U$  of the tank liquid heights with respect to inlet flow,

$$U_{Q_1}^{x_i}, \quad i = 1, 2.$$

The differential equations for the variation in tank height are

$$A_1 \frac{dx_1}{dt} = Q_1 - \frac{x_1 - x_2}{R_1} \quad (5.7)$$

$$A_2 \frac{dx_2}{dt} = \frac{x_1 - x_2}{R_1} - \frac{x_2}{R_2} \quad (5.8)$$

Since, by Eq. (5.3),

$$U_{Q_1}^{x_1} = \partial x_1 / \partial Q_1 = U_1 \quad \text{and} \quad U_2 = U_{Q_1}^{x_2} = \partial x_2 / \partial Q_1$$

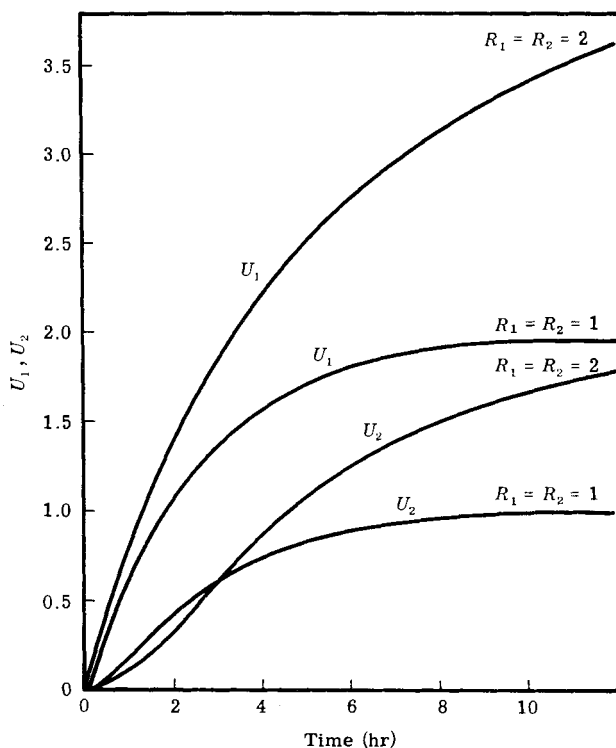


Fig. 5.2. Sensitivity coefficients versus time (two-tank system). (From Engelstadt [1, p. 28].)



we obtain

$$A_1 \frac{dU_1}{dt} = 1 - \frac{U_1 - U_2}{R_1} \quad (5.9)$$

$$A_2 \frac{dU_2}{dt} = \frac{U_1 - U_2}{R_1} - \frac{U_2}{R_2} \quad (5.10)$$

Setting  $A_1 = A_2 = 1$ , this pair of differential equations can be solved to give  $U_1$  and  $U_2$  as a function of  $t$ . Figure 5.2 shows these sensitivities as a function of time for two values of  $R_1$  and  $R_2$ . The curves for  $U_1$  are first-order exponentials, whereas second-order characteristics are exhibited by  $U_2$ .

### Derivation of Network Functions

When the system under consideration is represented in the form of a SFG, the value of any output  $x_i$  is computed via Eq. (5.11).

$$x_i = \sum_{j=1}^n (T_{j \rightarrow i})(\text{input}_j) \quad (5.11)$$

From Eq. (5.11) we deduce that the  $n$  transmittances  $T_{j \rightarrow i}$  constitute a set of parameters closely related to the value of the output  $x_i$ . If  $p$  is a system parameter subject to variation, its effect on output  $x_i$  can be determined by differentiating Eq. (5.11)

$$\partial x_i / \partial p = \sum_{j=1}^n (\partial T_{j \rightarrow i} / \partial p)(\text{input}_j) \quad (5.12)$$

The set of values  $(\partial T_{j \rightarrow i} / \partial p)$  are *sensitivities* and represent the variation in the transmittance  $T_{j \rightarrow i}$  with respect to variations in a system parameter  $p$ .

Often it is more desirable to speak of the percentage change in the performance criteria resulting from a percentage change in the system parameter. This is the *logarithmic sensitivity* and it is given by

$$S_p^T = (\partial \ln T_{j \rightarrow i}) / (\partial \ln p) = (\partial T_{j \rightarrow i} / \partial p) p / T_{j \rightarrow i} \quad (5.13)$$

In Eqs. (5.11) through (5.13), the transfer functions  $T_{j \rightarrow i}$  are, in the general case,  $T_{j \rightarrow i}(s, p)$ ; that is, they are functions of the Laplace operator  $s$  and some particular parameter  $p$ . If the input  $j$  and the output  $i$  are understood, the transfer function can be written as  $T(s, p)$ , which is obtainable via

Mason's rule as a ratio of two polynomials

$$T(s, p) = N(s, p)/D(s, p) \quad (5.14)$$

If we regard the branch gains as parameters, the numerator and denominator of this equation can be written in bilinear form to yield

$$T(s, p) = \frac{N(s)}{D(s)} = \frac{N_1(s) + pN_2(s)}{D_1(s) + pD_2(s)} \quad (5.15)$$

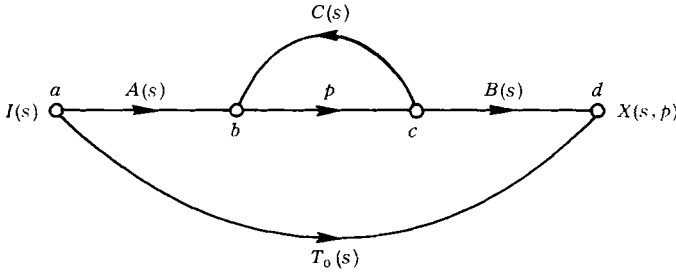


Fig. 5.3. Generalized SFG.

Figure 5.3 represents a signal flow graph of arbitrary complexity which has been reduced to a simplified form. The input of interest  $I(s)$ , the output of interest  $X(s, p)$ , and the branch corresponding to the parameter of interest  $p$  are preserved in the reduced graph. The functions  $A(s)$ ,  $B(s)$ , and  $C(s)$  are branch gains associated with the reduced graph. The function  $T_0(s)$  is the *leakage transmittance* and defines the relationship between input and output when the parameter  $p$  is set to zero. Applying Mason's rule to this graph yields

$$\begin{aligned} T(s, p) &= \frac{T_0(s)[1 - pC(s)] + pA(s)B(s)}{[1 - pC(s)]} \\ &= T_0(s) + \frac{pA(s)B(s)}{[1 - pC(s)]} \end{aligned} \quad (5.16)$$

From Eqs. (5.14) and (5.16), it is apparent that

$$T(s, 0) = T_0(s) = N_1(s)/D_1(s) \quad (5.17)$$

Substituting Eq. (5.17) into Eq. (5.16) yields

$$T(s, p) = \frac{N_1(s)[1 - pC(s)] + pA(s)B(s)D_1(s)}{D_1(s)[1 - pC(s)]} \quad (5.18)$$

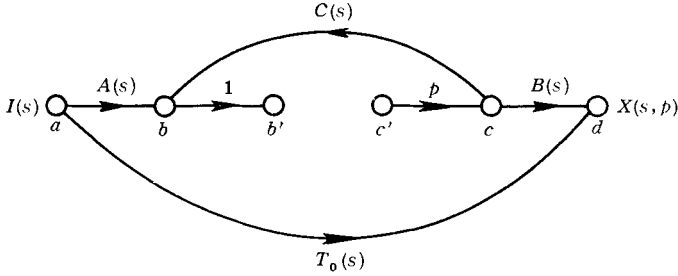


Fig. 5.4. SFG used in defining the return difference.

Equating the numerators and denominators of Eqs. (5.15) and (5.18) leads directly to expressions for  $C(s)$  and  $A(s)B(s)$ .

$$C(s) = -D_2(s)/D_1(s) \quad (5.19)$$

$$A(s) \cdot B(s) = [N_2(s)D_1(s) - N_1(s)D_2(s)]/D_1^2(s) \quad (5.20)$$

If branch  $bc$  in Fig. 5.3 is broken, as in Fig. 5.4, and a signal of unity is applied at node  $c'$ , the difference between the applied signal and that returned to node  $b'$  will be  $1 - pC(s)$ , provided  $I(s) = 0$ . The quantity  $1 - pC(s)$  is known as the *return difference*  $R_p(s)$ .

$$R_p(s) = 1 - pC(s) \quad (5.21)$$

Substituting Eq. (5.19) for  $C(s)$  in Eq. (5.21):

$$\begin{aligned} R_p(s) &= 1 + pD_2(s)/D_1(s) \\ &= [D_1(s) + pD_2(s)]/D_1(s) \\ &= D(s, p)/D_1(s) \end{aligned} \quad (5.22)$$

The return difference computed under the condition of zero output ( $X(s, p) = 0$ ) is the *null-return difference*,  $R_p^0(s)$ . The input signal required to obtain a zero output is  $-pB(s)/T_0(s)$ . The contribution of this input to the signal at  $b'$  is  $-pA(s) \cdot B(s)/T_0(s)$ , and the contribution of the unity input at  $c'$  is  $pC(s)$ . The null-return difference is then

$$\begin{aligned} R_p^0(s) &= 1 - [pC(s) - pA(s) \cdot B(s)/T_0(s)] \\ &= 1 - pC(s) + pA(s) \cdot B(s)/T_0(s) \end{aligned} \quad (5.23)$$

Substituting Eqs. (5.17), (5.19), and (5.20) into Eq. (5.23) yields

$$R_p^0 = 1 + pN_2(s)/N_1(s) = N(s, p)/N_1(s) \quad (5.24)$$

## Deriving Sensitivity Functions

Equation (5.16) expresses the transfer function  $T(s, p)$  in the most convenient form.

$$T(s, p) = T_0 + \frac{pA(s) \cdot B(s)}{1 - pC(s)} \quad (5.25)$$

Differentiating this function with respect to  $p$  yields the desired sensitivity

$$\frac{\partial T(s, p)}{\partial p} = \frac{A(s) \cdot B(s)}{[1 - pC(s)]^2} \quad (5.26)$$

The corresponding logarithmic sensitivity is obtained by multiplying  $\partial T(s, p)/\partial p$  by  $p/T(s, p)$ .

$$\begin{aligned} S_p^{T(s, p)} &= \frac{A(s) \cdot B(s)}{[1 - pC(s)]^2} \cdot \frac{p}{T(s, p)} \\ &= \frac{1}{R_p(s)} \left[ 1 - \frac{T_0}{T(s, p)} \right] \end{aligned} \quad (5.27)$$

Equation (5.27) can be further simplified by algebraic manipulation to yield

$$\begin{aligned} S_p^{T(s, p)} &= \frac{1}{R_p(s)} - \frac{1}{R_p^0(s)} \\ &= \frac{D_1(s)}{D(s, p)} - \frac{N_1(s)}{N(s, p)} \end{aligned} \quad (5.28)$$

Equation (5.28) indicates a direct means of evaluating the sensitivity function  $S_p^{T(s, p)}$  without performing differentiations.  $N_1(s)$  and  $D_1(s)$  can be evaluated by Eq. (5.17); that is, by using Mason's rule to evaluate the transfer function with the parameter  $p$  set to zero.  $N(s, p)$  and  $D(s, p)$  can be obtained by applying Mason's rule with the parameter set at its nominal value. When the leakage transmission  $T_0$  is negligible,  $N_1$  is negligible and Eq. (5.28) reduces to

$$S_p^{T(s, p)} = D_1(s)/D(s, p) = 1/R_p(s) \quad (5.29)$$

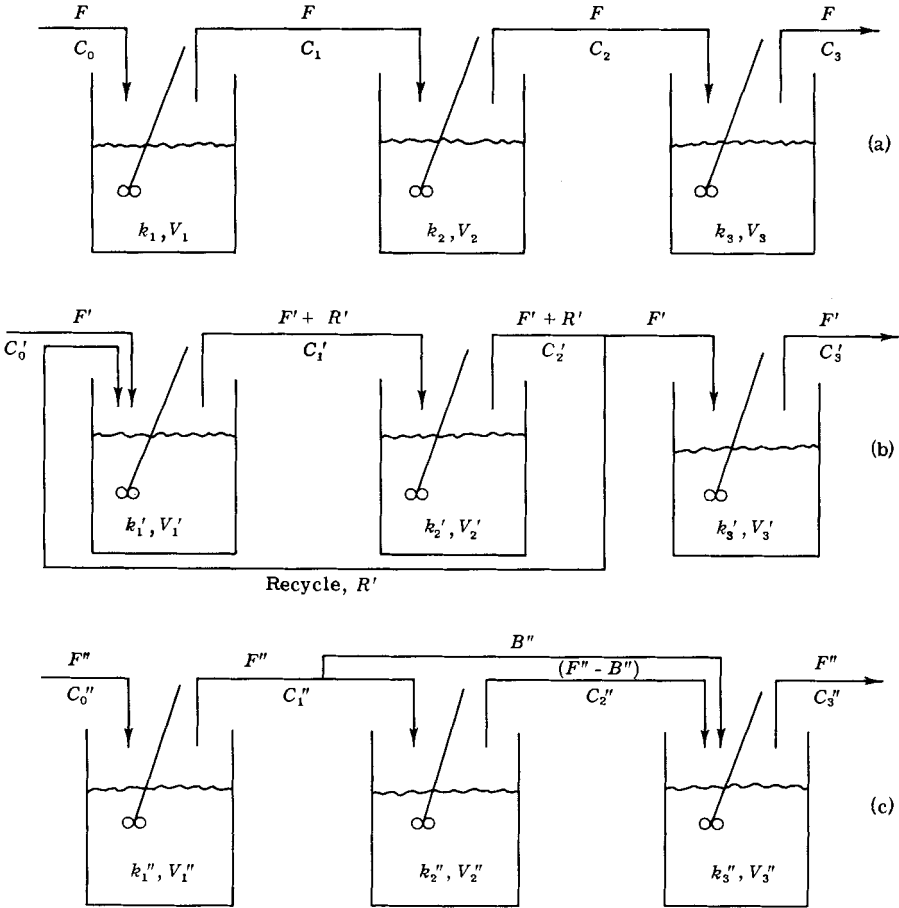
If the graph under consideration contains a number of branches having gains  $q_1(s, p)$  through  $q_N(s, p)$ , the sensitivity function can be evaluated as

$$S_p^{T(s, p)} = \frac{\partial \ln T(s, p)}{\partial \ln p} = \sum_{i=1}^N \frac{\partial \ln T(s, p)}{\partial \ln q_i(s, p)} \cdot \frac{\partial \ln q_i(s, p)}{\partial \ln p} \quad (5.30)$$

the derivatives of the transfer function  $T(s, p)$  are evaluated as previously described.

### Applying Sensitivity Functions to Structural Analysis

The application of sensitivity concepts is now demonstrated by an example from Engelstadt whose doctoral thesis has been referred earlier [1]. He poses the problem of how a designer might reduce the fluctuation of an output variable in the face of fluctuations in one of the system parameters.



**Fig. 5.5.** Stirred tank reactors with recycle and bypass. (a) Case I (no control). (b) Case II (recycle to reactor 1). (c) Case III (bypass around reactor 2).

The straw man in this analysis is the system of stirred tank reactors in Fig. 5.5. The  $k$ 's are first-order reaction rate constants in  $hr^{-1}$ ,  $F$ 's are flows ( $lb/ft^3$ ),  $V$ 's are tank volumes ( $ft^3$ ), and the  $C$ 's concentration ( $lb/ft^3$ ). In a stirred-tank cascade such as this, the outlet concentrations are equal to the concentrations in the tank. We seek to reduce the fluctuations in outlet concentration in the face of poor temperature control in the second reactor, which results in a fluctuating  $k_2$ . Our objective, therefore, is to minimize  $S_{a_2}^{C_3}$ , the sensitivity of the output variable  $C_3$  to  $a_2$ , where  $a_2 = f(k_2)$ .

Three flow sheet configurations are proposed in Fig. 5.5. System I is unmodified; in system II we introduce recycle from reactor 2 to 1, and in system III we bypass material around reactor 2. We consider only the steady state.

The material balances for the three cases are

Case I:

$$\begin{aligned} C_1 &= C_0 F / (F + k_1 V_1) = C_0 a_1 \\ C_2 &= C_1 F / (F + k_2 V_2) = C_1 a_2 \\ C_3 &= C_2 F / (F + k_3 V_3) = C_2 a_3 \end{aligned} \quad (5.31)$$

Case II:

$$\begin{aligned} C_1' &= \frac{C_0' F'}{(F' + R') + k_1' V_1'} + \frac{C_2' R'}{(F' + R') + k_1' V_1'} = C_0' a_1' + C_2' K' \\ C_2' &= \frac{C_1' (F' + R')}{(F' + R') + k_2' V_2'} = C_1' a_2' \\ C_3' &= \frac{C_2' F'}{F' + k_3' V_3'} = C_2' a_3' \end{aligned} \quad (5.32)$$

Case III:

$$\begin{aligned} C_1'' &= \frac{C_0'' F''}{F'' + k_1'' V_1''} = C_0'' a_1'' \\ C_2'' &= \frac{C_1'' (F'' - B'')}{(F'' - B'') + k_2'' V_2''} = C_1'' a_2'' \\ C_3'' &= \frac{C_2'' (F'' - B'')}{F'' + k_3'' V_3''} + \frac{C_1'' B''}{F'' + k_3'' V_3''} = C_2'' a_3'' + C_1'' K'' \end{aligned} \quad (5.33)$$

The flow chart corresponding to these equations and the sensitivities of the output node  $C_3$  are shown in Fig. 5.6.

The conclusions to be drawn from the sensitivities is that negative feedback, case II, increases the sensitivity, while bypass case III, desensitizes

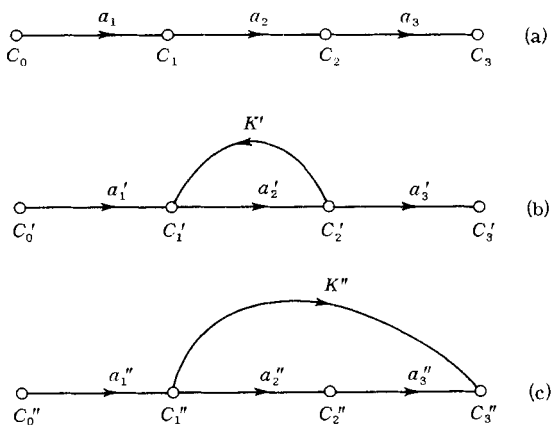


Fig. 5.6. Flow graphs and sensitivities for reactors. (a) Case I,  $S_{a_2}^T = 1$ . (b) Case II,  $S_{a_2}^T = (1 - a_2'K')^{-1}$ . (c) Case III,  $S_{a_2}^T = [1 + (K''/a_2'a_2'')]^{-1}$ .

the system. This is a perfectly obvious result considering that, if we completely “shortcircuit” reactor 2, the output is in no way dependent on fluctuations in  $k_2$  (or  $a_2 = f(k_2)$ ). Conversely, the greater the feedback, the more material passes through tank 2, which explains the increase in sensitivity for case II. We now focus attention on case II and show how the sensitivity can be obtained by use of Mason’s rule and previously derived equations.

1. By Eq. (5.13): There is one loop,  $L_1 = a_2'K'$  by Mason’s rule:

$$T_{C_0' \rightarrow C_3'} = (\sum P_k \Delta_k) / \Delta = a_1' a_2' a_3' / (1 - a_2' K') \quad (5.34)$$

since  $\Delta = 1 - a_2' K'$ ;  $P_k = a_1' a_2' a_3'$ ;  $\Delta_k = 1$ . Now, letting  $a_2' = p$

$$S_p^T = \frac{\partial T_{j \rightarrow i}}{\partial p} \frac{p}{T_{j \rightarrow i}} = \frac{\partial T_{C_0' \rightarrow C_3'}}{\partial a_2'} \frac{a_2'}{T_{C_0' \rightarrow C_3'}} = \frac{1}{1 - a_2' K'} \quad (5.35)$$

This confirms the result in Fig. 5.6. We now examine alternative methods of obtaining this result.

2. By Eq. (5.16): There is no leakage transmission, thus  $T_0(s) = 0$ .  $A(s) = a_1'$ ,  $B(s) = a_3'$ ,  $C(s) = K'$ , and

$$T_{C_0' \rightarrow C_3'} = T_0(s) + \frac{p A(s) B(s)}{1 - p C(s)} = \frac{a_1' a_2' a_3'}{1 - a_2' K'} \quad (5.36)$$

which agrees with the result previously obtained [Eq. (5.34)].

3. By Eq. (5.28), comparing Eq. (5.15) and (5.34) reveals that

$$N_1(s) = 0, \quad N_2(s) = a_1' a_3', \quad D_1(s) = 1, \quad D_2(s) = -K' \quad (5.37)$$

By Eq. (5.22)

$$R_p = [D_1(s) + pD_2(s)]/D_1(s) = D(s, p)/D_1(s) \quad (5.38)$$

Thus,

$$D(s, p) = 1 - a_2' K' \quad (5.39)$$

By Eq. (5.24),

$$R_p^0 = [1 + pN_2(s)]/N_1(s) = N(s, p)/N_1(s) \quad (5.40)$$

Thus  $N(s, p) = 1 + a_2' a_1' a_3'$ .

We now obtain the previous result using (5.28) since

$$S_p^T = \frac{D_1(s)}{D(s, p)} - \frac{N_1(s)}{N(s, p)} = \frac{1}{1 - a_2' K'} - \frac{0}{1 + a_2' a_1' a_3'} = \frac{1}{1 - a_2' K'} \quad (5.41)$$

or

$$S_p^T = \frac{1}{R_p(s)} - \frac{1}{R_p^0(s)} = \frac{1}{1 - a_2' K'} - \frac{0}{1 + a_2' a_1' a_3'} = \frac{1}{1 - a_2' K'} \quad (5.42)$$

Alternatively, we could have used Eq. (5.29), since there is no leakage transmission.

It must be emphasized again that all of these results depended on our being able to obtain transmittances by Mason's formula. In this simple example, it was easily done by inspection. Chapters 6 and 7 will present more realistic design and analysis problems where the computer programs developed in Chapter 3 must be used.

### Gain and Phase Sensitivity

Any logarithmic sensitivity function  $S_p^{T(s)}$  can be expressed in terms of complex frequency  $S_p^{T(j\omega)}$ , just as the transfer function  $T(s)$  was converted to  $T(j\omega)$  in the previous chapter.

$$S_p^{T(j\omega)} = [\partial \ln T(j\omega)] / \partial \ln p \quad (5.43)$$

The transfer function  $T(j\omega)$  can be written in logarithmic form as

$$\ln T(j\omega) = \alpha(\omega) + j\beta(\omega) \quad (5.44)$$



where  $\alpha(\omega)$  and  $\beta(\omega)$  are the *gain* and *phase functions*, respectively. The corresponding gain and phase sensitivities are defined as

$$S_p^{\alpha(\omega)} = \partial \alpha(\omega) / \partial \ln p \quad (5.45)$$

$$S_p^{\beta(\omega)} = \partial \beta(\omega) / \partial \ln p \quad (5.46)$$

Substituting Eq. (5.44) into Eq. (5.43) and differentiating yields

$$\begin{aligned} S_p^{T(j\omega)} &= \frac{\partial \ln T(j\omega)}{\partial \ln p} = \frac{\partial [\alpha(\omega) + j\beta(\omega)]}{\partial p/p} \\ &= \frac{\partial \alpha(\omega)}{\partial p/p} + j \frac{\partial \beta(\omega)}{\partial p/p} \end{aligned} \quad (5.47)$$

Equation (5.47) indicates that the gain and phase sensitivities are given by the real and imaginary parts of  $S_p^{T(j\omega)}$ , respectively. This result is useful in determining the effect of parameter variations on a system's frequency response characteristics.

### Pole and Zero Sensitivity

The nature of a system's transient response is determined by the location of the poles and zeros of the appropriate transfer function. For this reason, it is often useful to know the shifts in the locations of poles and zeros which result from variations in the system parameters. It has been shown that the shift in a pole  $p_i$ , of multiplicity  $m$ , caused by a variation  $\delta p$  in a parameter  $p$ , is given by

$$\delta p_i = \left[ L_m \left( \frac{\delta p}{p} \right) \right]^{1/m} \quad (5.48)$$

where

$$L_m = \frac{(s - p_i)^m}{R_p(s)} \bigg|_{s=p_i} \quad (5.49)$$

Likewise, the shift in a zero  $Z_i$  of multiplicity  $m$  caused by a variation  $\delta p$  in parameter  $p$  is given by

$$\delta Z_i = [L'_m (\delta p/p)]^{1/m} \quad (5.50)$$

where

$$L'_m = \frac{(s - Z_i)^m}{R_p^0(s)} \bigg|_{s=Z_i} \quad (5.51)$$

## Sensitivities via Structural Methods

Equation (5.26), which was obtained by differentiating Eq. (5.25), is an expression for the sensitivity  $\partial T(s, p)/\partial p$ . The physical interpretation of this equation can be seen by writing it in a slightly different form.

$$\frac{\partial T(s, p)}{\partial p} = \frac{\overbrace{A(s)}^{T_{a \rightarrow b}}}{[1 - pC(s)]} \cdot \frac{\overbrace{B(s)}^{T_{f \rightarrow d}}}{[1 - pC(s)]} \quad (5.52)$$

With reference to Fig. 5.7, Eq. (5.52) is seen to be made up of the product of two transfer functions. The first is that from node  $a$  to node  $b$ , and the second transfer function is that from a fictitious input node  $f$  entering  $c$  to the output node  $d$ . These two transfer functions are easily verified by applying Mason's rule.

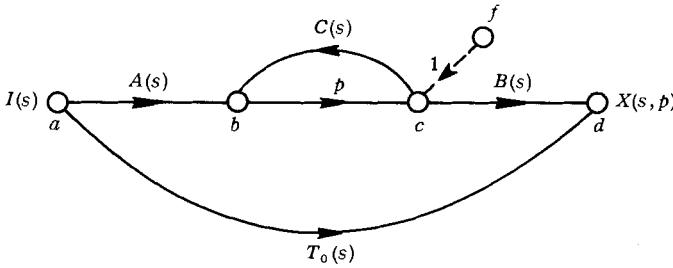


Fig. 5.7. SFG formulation for the computation of  $\partial T/\partial p$ .

Equation (5.52) is important because it provides a simple means of computing the sensitivity  $\partial T(s, p)/\partial p$ . The procedure requires only the evaluation of two transfer functions and their subsequent multiplication to yield  $\partial T(s, p)/\partial p$ . The logarithmic sensitivity can be obtained by multiplying the sensitivity  $\partial T(s, p)/\partial p$  by  $p/T(s, p)$ .

$$S_p^T = \frac{\partial \ln T(s, p)}{\partial p} = \frac{\partial T(s, p)}{\partial p} \cdot \frac{p}{T(s, p)} \quad (5.53)$$

This technique applies equally well to the determination of sensitivities with respect to a function  $q$ , where  $q$  is a branch gain and a function of the particular parameter of interest  $p$ ,  $q = f(s, p)$ . The ultimate sensitivity is then obtained as

$$\frac{\partial T(s, p)}{\partial p} = \frac{\partial T(s, p)}{\partial q(s, p)} \cdot \frac{\partial q(s, p)}{\partial p} \quad (5.54)$$

When  $N$  branches exist which have gains  $q_1(s, p)$  through  $q_N(s, p)$ , the ultimate sensitivity is given by

$$\frac{\partial T(s, p)}{\partial p} = \sum_{i=1}^N \frac{\partial T(s, p)}{\partial q_i(s, p)} \cdot \frac{\partial q_i(s, p)}{\partial p} \quad (5.55)$$

Equation (5.55) may be used to compute the sensitivity functions when the parameter appears in several locations within a SFG.

The sensitivity of an output variable with respect to a given parameter can be obtained by applying Eq. (5.12)

$$\frac{\partial X_i}{\partial p} = \sum_{j=1}^N \frac{\partial T_{j \rightarrow i}(s, p)}{\partial p} \cdot I_j(s) \quad (5.56)$$

where the necessary derivatives,  $\partial T_{j \rightarrow i}(s)/\partial p$ , are obtained from Eq. (5.55).

A more direct method of obtaining the same information is available, the technique being known as the *method of varied branch transmittances*. Figure 5.8 displays a generalized signal flow graph such as that considered previously, and a second graph identical to the first. The branch  $bc$ , which has the parameter  $p$  as its gain, can be bypassed by connecting the two graphs by the branch  $bc'$ . The transfer function relating  $I(s)$  to  $X'(s, p)$  is obtained by applying Mason's rule to the system.

$$T(s, p) = \frac{A(s) \cdot B(s)}{[1 - 2pC(s) + p^2C^2(s)]} \quad (5.57)$$

$$T(s, p) = \frac{A(s) \cdot B(s)}{[1 - pC(s)]^2}$$

This transfer function is identical to Eq. (5.26), that is, the transfer function relating  $I(s)$  to  $X'(s, p)$  is identical to the derivative with respect to  $p$ , of the transfer function relating  $I(s)$  to  $X(s, p)$ . It is therefore possible to obtain the signal  $\partial X(s, p)/\partial p$  at node  $d'$  in the auxiliary graph.

If the branch  $bc$  has a gain which is a function of  $p$ ,  $q(s, p)$ , the bypass branch  $bc'$  is given a gain having the value  $\partial q(s, p)/\partial p$ . This has the effect of multiplying the transfer function relating  $I(s)$  to  $X'(s)$  by  $\partial q(s, p)/\partial p$  and the resulting output will be

$$\frac{\partial X(s, p)}{\partial q(s, p)} \cdot \frac{\partial q(s, p)}{\partial p} = \frac{\partial X(s, p)}{\partial p} \quad (5.58)$$

Any number of branches having gains  $q_1(s, p)$  through  $q_N(s, p)$  can be

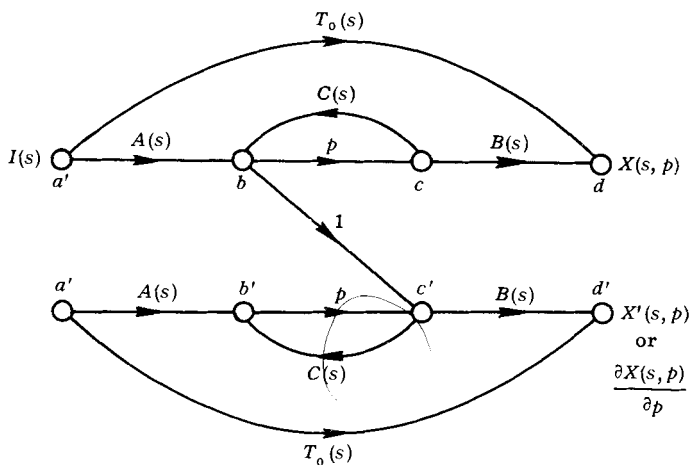


Fig. 5.8. Method of varied branch transmittances, SFG formulation.

treated simultaneously. Each bypass branch provides a unique path from the input to the output and the result obtained by Mason's rule is essentially

$$\frac{\partial T(s, p)}{\partial p} = \sum_{i=1}^N \frac{\partial T(s, p)}{\partial q_i(s, p)} \cdot \frac{\partial q_i(s, p)}{\partial p} \quad (5.59)$$

The method of varied branch transmittances is beautiful from the standpoint of simplicity, but it leads to computational difficulties. Since the technique requires two identical graphs, the dimensionality of the original model is doubled. The loops in each graph are distinct and none of the loops in the first graph touch any of those in the second. This leads to very lengthy computations in evaluating the network functions when the graphs are large and have many loops.

Another structural method of determining sensitivities is the *sensitivity points method*. This technique is applicable to many systems with one input and one output and yields the sensitivities  $\partial X(s, p_1, \dots, p_m) / \partial \ln p_i$  in situ. In applying this technique, the output signal of the system is introduced as the input signal to an identical auxiliary graph, and the desired sensitivities  $\partial X(s, p_1, \dots, p_m) / \partial \ln p_i$  appear at locations in the auxiliary graph called *sensitivity points*. Determining the location of the various sensitivity points is the most difficult part of applying the technique. Tomovic [2] discusses the method and its implementation on analog computers. Sedlar [3] also discusses it and has contributed some interesting ideas pertaining to its extension to systems of arbitrary complexity.

### Sensitivities for Large Parameter Variations

The techniques discussed in the preceding sections yield results which are rigorously valid only in the case of small, or differential, parameter variations. Care must be taken when using the results to predict variations in transfer functions or outputs caused by large parameter variations.

If  $p$  represents the nominal value of some parameter and  $\Delta p$  represents a deviation from this nominal value, then

$$T(s, p + \Delta p) = T(s, p) + \Delta T \quad (5.60)$$

Substituting (5.16) for  $T(s, p + \Delta p)$  and  $T(s, p)$  in the above equation yields

$$\Delta T = \frac{A(s)B(s)\Delta p}{[1 - (p + \Delta p)C(s)][1 - pC(s)]} \quad (5.61)$$

Substituting

$$T(s, p) - T_0(s) = \frac{pA(s)B(s)}{1 - pC(s)}$$

and further rearrangement yields

$$\frac{\Delta T/[T(s, p) - T_0(s)]}{\Delta p/p} = \frac{1}{1 - (p + \Delta p)C(s)} = \frac{1}{R_{(p+\Delta p)}} \quad (5.62)$$

Equation (5.62) defines the sensitivity of  $T(s, p)$  with respect to large variations in the parameter  $p$ . When the leakage transmission is zero, this expression is very similar to the expression for the sensitivity due to differential parameter variations, Eq. (5.29).

### Transfer Functions and Sensitivities in Static Systems

For dynamic systems, the desired transfer functions are obtained as a ratio of two polynomials in  $s$  by means of the techniques developed in Chapter 3. In the case of static systems, where the Laplace variable  $s$  does not appear, one parameter, or some grouping of parameters  $\beta$  can be retained in symbolic form and the transfer function is then obtained as a ratio of two polynomials  $\beta$ .

$$T(\beta) = N(\beta)/D(\beta) \quad (5.63)$$

This capability is useful since it provides a means of obtaining closed form

expressions for a system's outputs in terms of some parameter or grouping of parameters.

$$x_i(\beta) = \sum_{j=1}^N T_{j \rightarrow i}(\beta) \cdot I_j \quad (5.64)$$

Once the solution is obtained in closed form, the output can be evaluated for any value of the parameter and the sensitivities can be obtained by direct differentiation.

$$S_p^{T(p)} = \frac{\partial T(p)}{\partial p} \cdot \frac{p}{T} \quad (5.65)$$

$$S_p^{x(p)} = \frac{\partial x(p)}{\partial p} \cdot \frac{p}{x(p)} \quad (5.66)$$

The material discussed in this chapter covers only the basic and more powerful techniques of sensitivity analysis. More detailed discussions, which include many related topics, are available in the literature [4-6].

## REFERENCES

1. D. G. Engelstadt, On decision variable contingency and system desensitization in process design. Ph. D. Dissertation, Univ. of Wisconsin, Madison, Wisconsin, 1970.
2. R. Tomovic, "Sensitivity Analysis of Dynamic Systems." McGraw-Hill, New York, 1964.
3. M. Sedlar, On the theory of signal flow graphs. Tech. Rep. USCEE-300. Univ. of Southern California, Santa Monica, California, August 1968.
4. S. K. Mitra, "Analysis and Synthesis of Linear Active Networks." Wiley, New York, 1969.
5. R. M. Munoz and S. Park Chan, The transfer function approach to sensitivity analysis by digital computer. *Proc. Midwest. Symp. Circuit Theory, 11th, Univ. of Notre Dame, Notre Dame, Indiana, May 13-14, 1968*, p. 558.
6. L. Radanovic, ed., *Proc. Int. Symp. on Sensitivity Anal., Dubrovnik, Yugoslavia, 1964*. Symp. Publ. Div., Pergamon Press, Oxford, 1966.

This page intentionally left blank

## CHAPTER 6

### EXAMPLES AND FURTHER APPLICATIONS

In this chapter we present a number of examples which clearly illustrate the effectiveness and diversified utility of the flow-graph analysis techniques developed in the previous chapters. A number of problems are included which illustrate important applications not discussed thus far. These examples demonstrate the method of organizing problems when the subroutines of Appendix I are used, and illustrate the types and format of outputs obtainable.

#### Closed-Form Solution of an Absorption Column

The absorption column in Fig. 6.1 will be considered as an example of the analysis of a static system.  $G$  pounds per hour of dry gas containing  $y_7$  pounds of contaminant per pound of dry gas are treated with  $L$  pounds per hour of solvent containing  $x_0$  pounds of contaminant per pound of solvent. We wish to obtain a closed form solution which gives the steady-state outlet compositions,  $x_6$  and  $y_1$ , as a function of the liquid to gas ratio  $L/G$  and the inlet compositions. The same technique will also be used to



obtain a closed form solution in terms of the inlet compositions and the Henry constant  $m$  used in computing the vapor-liquid equilibria.

A steady state material balance for the contaminant on the  $n$ th tray yields

$$Lx_{n-1} + Gy_{n+1} - Lx_n - Gy_n = 0 \quad (6.1)$$

Equation (6.1) represents six equations,  $0 < n < 7$ ,  $n = 1, 2, \dots, 6$ , and each of these equations can be solved for  $x_n$ .

$$x_n = x_{n-1} + (G/L)y_{n+1} - (G/L)y_n \quad (6.2)$$

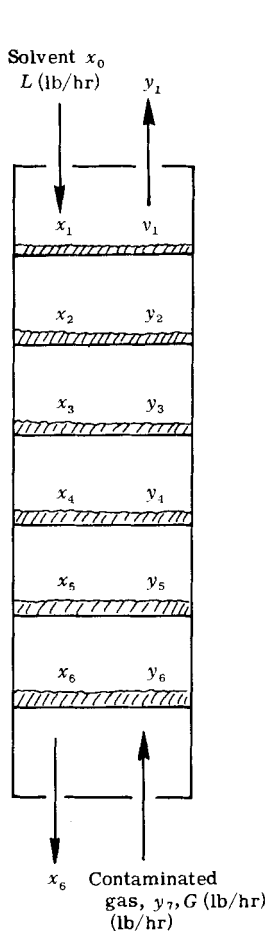


Fig. 6.1. Absorption column.

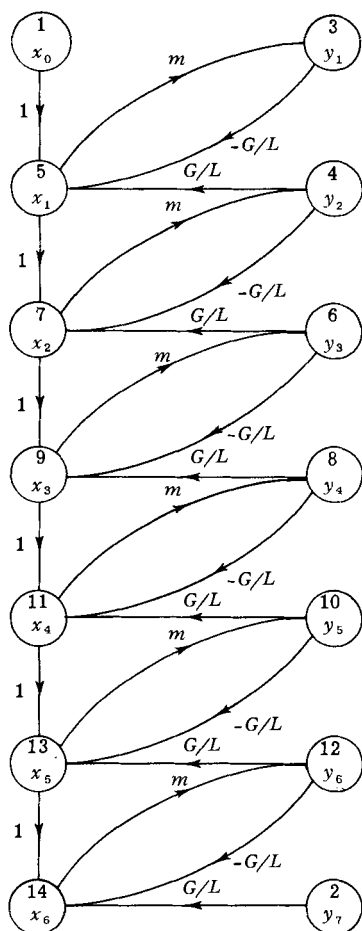


Fig. 6.2. SFG representation of Eqs. (6.2) and (6.3).

An expression for  $y_n$  follows immediately from Henry's law.

$$y_n = mx_n \quad (6.3)$$

Equations (6.2) and (6.3) can be cast in the form of the signal flow graph in Fig. 6.2.

The desired functions are now generated by using Mason's rule and the subroutines presented in Appendix I. To obtain the results as a function of  $L/G$ , it is convenient to handle this variable like the Laplace operator so we give it the symbol  $s$ . Table 6.1 shows the inventory of branches for the

TABLE 6.1  
INVENTORY OF BRANCHES AND GAINS CORRESPONDING TO FIG. 6.2 WHEN  $L/G$   
IS TREATED AS A SYMBOL

BRANCH NO.	ORIGIN	END	BRANCH GAIN	POWER OF $s$
1	1	5	1.00000	0
2	2	14	1.00000	-1
3	3	5	-1.00000	-1
4	4	5	1.00000	-1
5	4	7	-1.00000	-1
6	5	7	1.00000	0
7	6	7	1.00000	-1
8	6	9	-1.00000	-1
9	7	9	1.00000	0
10	8	9	1.00000	-1
11	8	11	-1.00000	-1
12	9	11	1.00000	0
13	10	11	1.00000	-1
14	10	13	-1.00000	-1
15	11	3	1.00000	0
16	12	13	1.00000	-1
17	12	14	-1.00000	-1
18	13	14	1.00000	0
19	14	12	0.700000	0
20	13	10	0.700000	0
21	14	8	0.700000	0
22	9	6	0.700000	0
23	7	4	0.700000	0
24	5	3	0.700000	0

SFG. Note that each branch has an *origin* node, an *end* node, and a numerical value plus a power of  $s$  ( $L/G$ ) for its gain. Since it is  $G/L$ , and not  $L/G$  that appears in the graph, the corresponding powers of  $L/G$  are  $-1$ . A nominal value for  $m$ , 0.7, is used for those branches having  $m$  as their gain, and the corresponding powers of  $L/G$  are, of course, zero. Since  $y_1$  corresponds to output node 3, and the inputs  $x_0$  and  $y_7$  correspond to nodes 1 and 2, respectively,  $y_1$  is given by

$$y_1 = T_{1 \rightarrow 3} \cdot x_0 + T_{2 \rightarrow 3} \cdot y_7 \quad (6.4)$$

The other output of interest  $x_6$  corresponds to node 14, and is given by

$$x_6 = T_{1 \rightarrow 14} \cdot x_0 + T_{2 \rightarrow 14} \cdot y_7 \quad (6.5)$$

Figure 6.3 is a schematic diagram of the method used to obtain the functional form of the required transmittances. Table 6.2 shows the computer

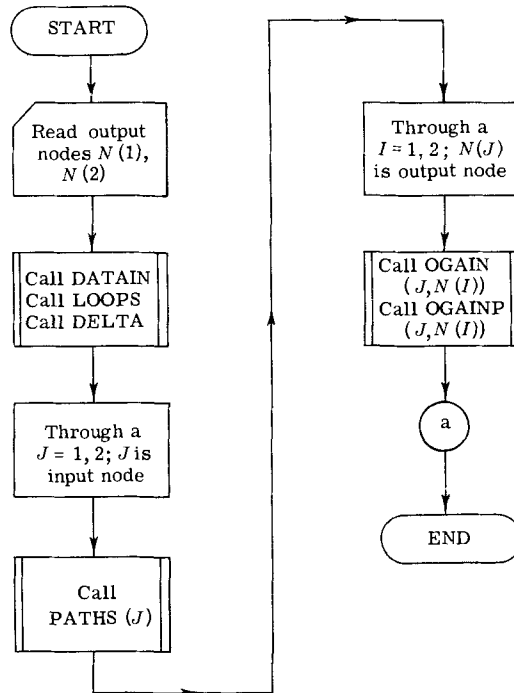


Fig. 6.3. Schematic diagram of a method to obtain the transmittances required in Eqs. (6.4) and (6.5).

output. Each transmittance is obtained as a ratio of two polynomials in  $L/G$ , the denominator polynomial  $\Delta$  being the same for each transfer function.

$$\begin{aligned} \Delta = & 0.118 + 0.168(L/G) + 0.24(L/G)^2 \\ & + 0.343(L/G)^3 + 0.49(L/G)^4 \\ & + 0.7(L/G)^5 + 1.0(L/G)^6 \end{aligned} \quad (6.6)$$

Substituting the expressions for the transfer functions into Eqs. (6.4) and

TABLE 6.2  
FUNCTIONAL FORM OF TRANSFER FUNCTIONS AS REQUIRED BY  
EQS. (6.4) AND (6.5),  $L/G = s$

FROM NODE 1 TO NODE 3		
POWER OF $s$	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.0	0.11764890
1	0.11764890	0.16806990
2	0.16806990	0.24009991
3	0.24009991	0.34299994
4	0.34299994	0.48999995
5	0.48999995	0.69999999
6	0.69999999	1.00000000

FROM NODE 2 TO NODE 3		
POWER OF $s$	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.11764890	0.11764890
1	0.0	0.16806990
2	0.0	0.24009991
3	0.0	0.34299994
4	0.0	0.48999995
5	0.0	0.69999999
6	0.0	1.00000000

FROM NODE 1 TO NODE 14		
POWER OF $s$	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.0	0.11764890
1	0.0	0.16806990
2	0.0	0.24009991
3	0.0	0.34299994
4	0.0	0.48999995
5	0.0	0.69999999
6	1.00000000	1.00000000

FROM NODE 2 TO NODE 14		
POWER OF $s$	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.16806990	0.11764890
1	0.24009991	0.16806990
2	0.34299994	0.24009991
3	0.48999995	0.34299994
4	0.69999999	0.48999995
5	1.00000000	0.69999999
6	0.0	1.00000000

(6.5), we obtain the desired expressions.

$$y_1 = \left[ \frac{0.118(L/G) + 0.168(L/G)^2 + 0.24(L/G)^3 + 0.343(L/G)^4 + 0.49(L/G)^5 + 0.7(L/G)^6}{\Delta} \right] x_0 + \left[ \frac{0.117}{\Delta} \right] y_7 \quad (6.7)$$

$$x_6 = \left[ \frac{(L/G)^6}{\Delta} \right] x_0 + \left[ \frac{0.168 + 0.24(L/G) + 0.343(L/G)^3 + 0.49(L/G)^4 + 0.7(L/G)^4 + (L/G)^5}{\Delta} \right] y_7 \quad (6.8)$$

The liquid and gas phase concentration profiles throughout the column can be determined by evaluating the transfer functions from the input nodes to the various intermediate nodes and substituting them in equations analogous to Eqs. (6.4) and (6.5).

Table 6.3 shows the branches and the associated gains corresponding to the case where  $m$  is treated symbolically and  $G/L$  is assigned a value of 1.6. Table 6.4 shows the transfer functions computed. These transfer functions can be substituted in Eqs. (6.4) and (6.5) to yield closed-form expressions

TABLE 6.3  
INVENTORY OF BRANCHES AND GAINS CORRESPONDING TO FIG. 6.2 WHEN  $m$   
IS TREATED AS A SYMBOL

ROW OF P.	ORIGIN	END	BRANCH GAIN	POWER OF S
1	1	5	1.00000	0
2	2	14	1.60000	0
3	3	5	-1.60000	0
4	4	5	1.60000	0
5	4	7	-1.60000	0
6	5	7	1.00000	0
7	6	7	1.60000	0
8	6	9	-1.60000	0
9	7	9	1.00000	0
10	8	9	1.60000	0
11	8	11	-1.60000	0
12	9	11	1.00000	0
13	10	11	1.60000	0
14	10	13	-1.60000	0
15	11	13	1.00000	0
16	12	13	1.60000	0
17	12	14	-1.60000	0
18	13	14	1.00000	0
19	14	12	1.00000	1
20	13	10	1.00000	1
21	11	8	1.00000	1
22	9	6	1.00000	1
23	7	4	1.00000	1
24	5	3	1.00000	1

TABLE 6.4  
FUNCTIONAL FORM OF TRANSFER FUNCTIONS AS REQUIRED BY  
EQS. (6.4) AND (6.5),  $m = s$

FROM NODE 1 TO NODE 3		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.0	1.0000000
1	1.0000000	1.5999994
2	1.5999994	2.5599976
3	2.5599976	4.0959940
4	4.0959940	6.5535879
5	6.5535879	10.485737
6	10.485737	16.777161

FROM NODE 2 TO NODE 3		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.0	1.0000000
1	0.0	1.5999994
2	0.0	2.5599976
3	0.0	4.0959940
4	0.0	6.5535879
5	0.0	10.485737
6	16.777161	16.777161

FROM NODE 1 TO NODE 14		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	1.0000000	1.0000000
1	0.0	1.5999994
2	0.0	2.5599976
3	0.0	4.0959940
4	0.0	6.5535879
5	0.0	10.485737
6	0.0	16.777161

FROM NODE 2 TO NODE 14		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	1.5999994	1.0000000
1	2.5599976	1.5999994
2	4.0959940	2.5599976
3	6.5535879	4.0959940
4	10.485737	6.5535879
5	16.777161	10.485737
6	0.0	16.777161

for the outlet compositions in terms of  $m$ . Differentiation of these functions with respect to  $m$  yields closed form expressions for the sensitivity of the outlet compositions with respect to  $m$ .

### Sensitivity Analysis of a Heat Exchanger Network

Consider the network of nine heat exchangers shown in Fig. 6.4. The optimum design of this system was obtained by Takamatsu [1] using the multilevel technique.<sup>†</sup> The object is to determine the sensitivity of the various outlet temperatures to a change in the heat transfer coefficient of any exchanger. This information is valuable in establishing area safety factors for the heat exchangers such that changes in the heat transfer coefficients due to fouling will not cause the outlet temperatures to differ from the desired values.

The symbols used in Fig. 6.4 have the following meanings:  $A$  = heat exchanger area, ft<sup>2</sup>;  $R$  = cold stream entry temperature, °F;  $S$  = cold stream exit temperature, °F;  $T$  = hot stream temperature, °F;  $UG$  = combination heat transfer coefficient and geometrical factor, Btu/ft<sup>2</sup>/hr/°F;  $wc$  = heat flow rate of a cold stream, Btu/hr/°F; and  $WC$  = heat flow rate of a hot stream, Btu/hr/°F. When subscripts appear, they indicate specific streams and/or units.

Two heat balances, one for the hot stream and one for the cold stream, can be written for each unit to yield a system of equations describing the network. A linear form of the average temperature driving force can be used since there are no drastic temperature changes through any unit. The average temperature driving force differs from the logarithmic mean temperature driving force by 2.002 degrees for the most severe case (unit 41). Considering unit 11, the two equations relating the stream temperatures are

$$(S_{11} - R_{11}) = \frac{UG_{11}A_{11}}{wc_{11}} \cdot \left[ \frac{(T_{11} - R_{11}) + (T_{10} - S_{11})}{2.0} \right] \quad (6.9)$$

$$(T_{10} - T_{11}) = \frac{UG_{11}A_{11}}{2WC_{11}} [(T_{11} - R_{11}) + (T_{10} - S_{11})] \quad (6.10)$$

Similar equations can be written for all units, and each equation can be solved for one of the intermediate or outlet stream temperatures.

<sup>†</sup> In Chapter 7 we show that the results obtained by Takamatsu are not optimal.

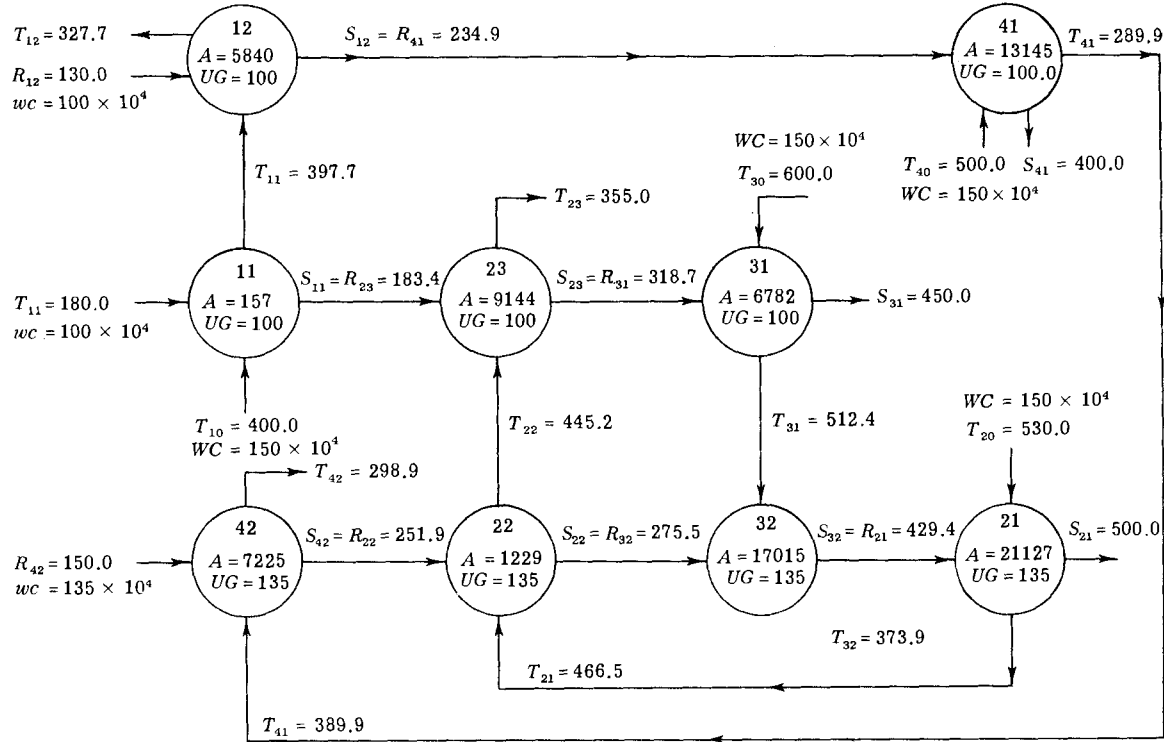


Fig. 6.4. Heat exchanger network.



For unit 11,

$$S_{11} = \left( \frac{\alpha_{11}}{1 + \alpha_{11}} \right) \cdot (T_{10} + T_{11}) + \left( \frac{1 - \alpha_{11}}{1 + \alpha_{11}} \right) R_{11} \quad (6.11)$$

$$T_{11} = \left( \frac{\beta_{11}}{1 + \beta_{11}} \right) \cdot (S_{11} + R_{11}) + \left( \frac{1 - \beta_{11}}{1 + \beta_{11}} \right) T_{10} \quad (6.12)$$

where

$$\alpha_{11} = (UG_{11} \cdot A_{11})/2wc_{11}, \quad \beta_{11} = (UG_{11} \cdot A_{11})/2WC_{11} \quad (6.13)$$

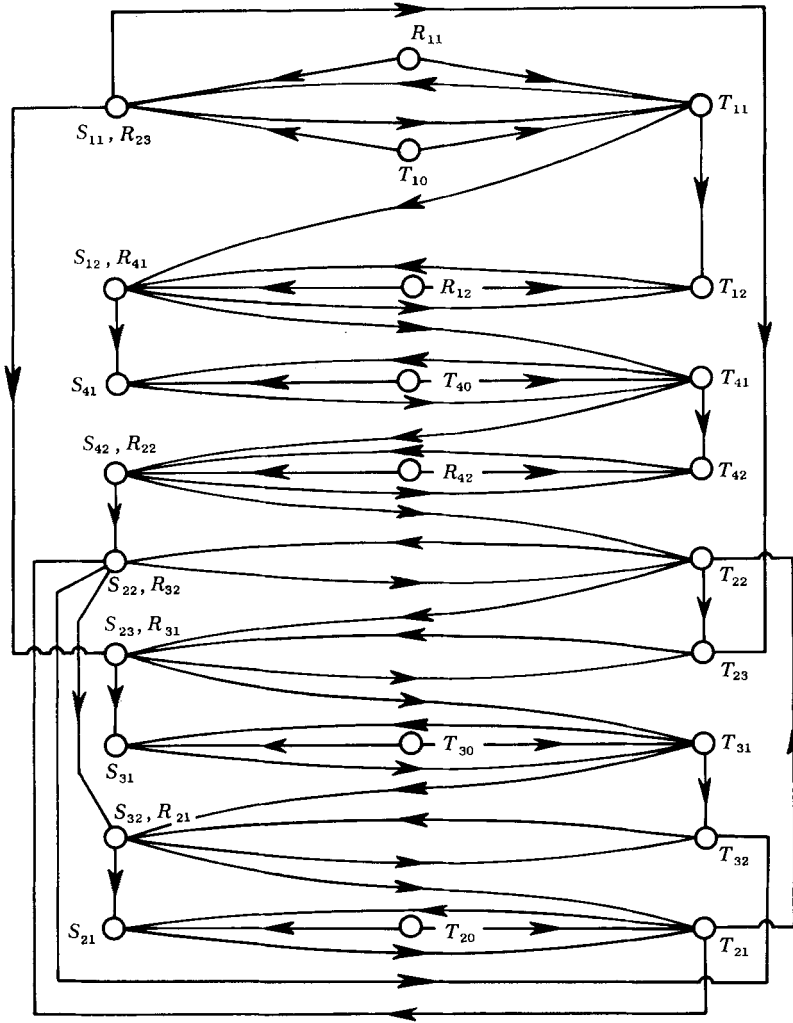


Fig. 6.5. SFG representation of heat transfer equations (gains not shown).

The SFG representation of the sets of equations is shown in Fig. 6.5. The graph contains one node for each stream temperature, seven of which are input nodes. The branch gains are not shown in Fig. 6.5 because of limited space; they are in each case a function of  $\alpha_{ij}$  and  $\beta_{ij}$  as indicated by Eqs. (6.12) and (6.13).

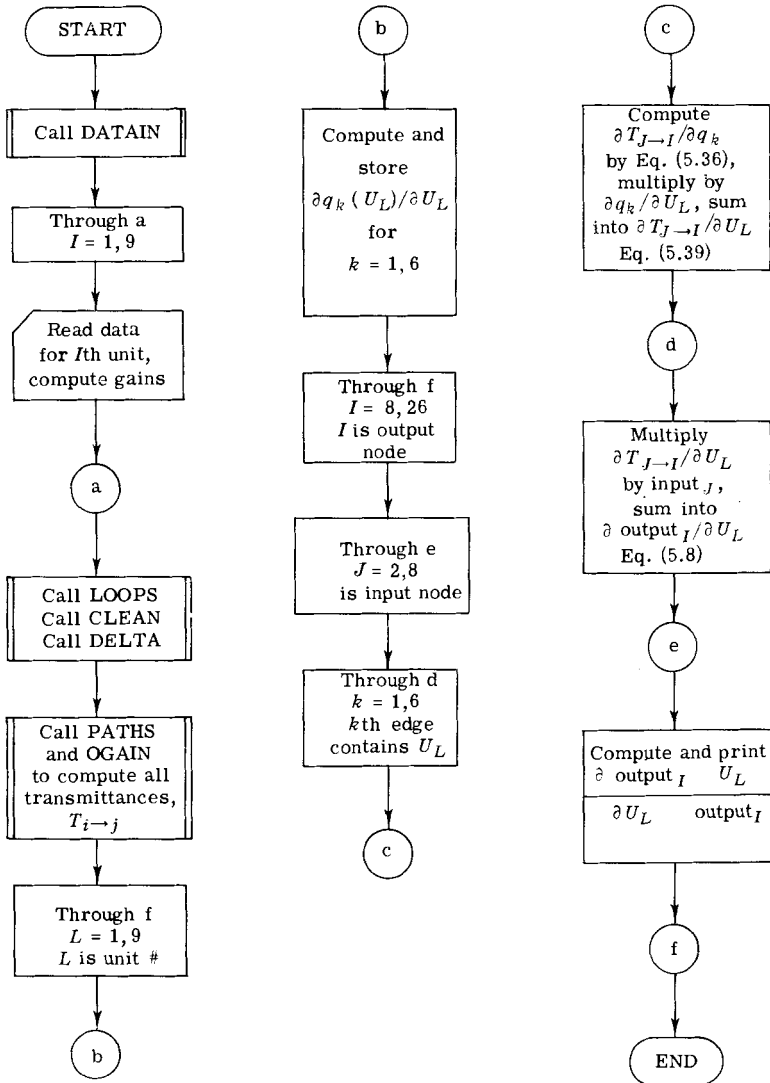


Fig. 6.6. Schematic diagram of an algorithm to compute the sensitivities for the heat exchanger network.

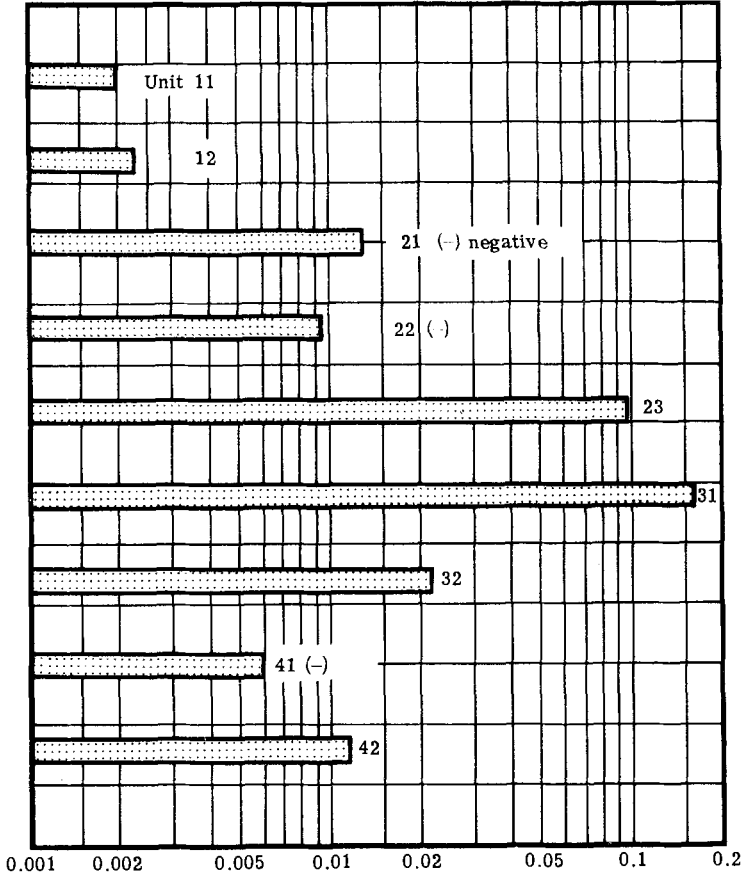


Fig. 6.7. Percent change in  $S_{31}$  per percent change in the heat transfer coefficient of each unit.

The logarithmic sensitivities of the various stream temperatures with respect to the various heat transfer coefficients are found by applying Eqs. (5.12), (5.52), (5.54), and (5.55). All of the necessary transmittances are obtained via the subroutines presented in Appendix I. Figure 6.6 is a schematic diagram of the technique used to compute the desired values. As an example of the results obtained, Fig. 6.7 shows both qualitatively and quantitatively the effect of changes in the various heat transfer coefficients on stream  $S_{31}$ . Table 6.5, row 2, shows the variation in the stream temperatures  $S_{21}$ ,  $S_{31}$ , and  $S_{41}$  resulting from a ten percent reduction in  $UG_{31}$ —as computed by the SFG method. These sensitivities are in excellent agreement

TABLE 6.5  
CHANGE IN STREAM TEMPERATURES RESULTING FROM A 10% DECREASE  
IN  $UG_{31}$  AS COMPUTED BY THREE DIFFERENT METHODS

$\Delta S_{ij}$			
Computation method	$\Delta S_{31}$	$\Delta S_{21}$	$\Delta S_{41}$
Direct computation	$-7.68^\circ$	$1.17^\circ$	$0.0^\circ$
Signal-flow-graph method	$-7.40^\circ$	$1.13^\circ$	$0.0^\circ$
Multilevel technique <sup>a</sup>	$-4.06^\circ$	$0.04^\circ$	$0.0^\circ$

<sup>a</sup> Data from [1].

with the results obtained by direct computation (row 1) and they agree reasonably well with those obtained by Takamatsu (row 3).

### Continuous Stirred Tank Reactor

As an example of the simulation and analysis of a dynamic system, the continuous stirred tank reactor system pictured in Fig. 6.8 will be analyzed. A solution containing  $C_{A_0}$  pounds of reactant  $A$  per gallon flows into the reactor at  $Q$  gallons per minute. In the reactor of volume  $V$ ,  $A$  is consumed by a second-order reaction having a rate constant  $k$ . The reactor effluent contains  $C_A$  and  $C_B$  pounds per gallon of species  $A$  and  $B$ , respectively. The concentration of  $B$  in the effluent is measured and relayed to a lead-lag controller which compares the measured value of  $C_B$  with the desired value  $C_{B\text{set}}$  and adjusts the flow rate of pure  $A$ ,  $Q_A$ , in order to correct for the deviation. This process involves a transportation lag, or dead time  $\tau$  since the measurement and information relay are not instantaneous. The desired steady-state operating conditions and parameter values are as follows:

$$\begin{aligned}
 \bar{C}_{A_0} &= 0.40 \text{ lb/gal}, & V &= 600.0 \text{ gal} \\
 \bar{Q} &= 50.0 \text{ gal/min}, & \varrho_A &= 9.04 \text{ lb/ft}^3 \\
 \bar{C}_A &= 0.84 \text{ lb/gal}, & k &= 0.1418 \text{ gal/lb/min} \\
 \bar{C}_B &= 1.0 \text{ lb/gal}, & \tau &= \frac{2}{3} \text{ min} \\
 \bar{Q}_A &= 10.0 \text{ gal/min}
 \end{aligned}$$

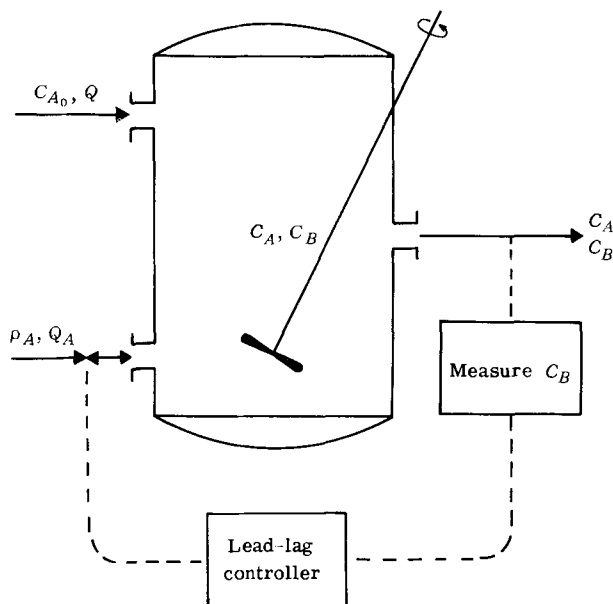


Fig. 6.8. Continuously stirred tank reactor with lead-lag controller.

Material balances on components  $A$  and  $B$  yield the differential equations which describe the reactor.

$$QC_{A_0} + \rho_A Q_A - C_A(Q + Q_A) - kC_A^2V = (dC_A/dt)V \quad (6.14)$$

$$- C_B(Q + Q_A) + kC_A^2V = (dC_B/dt)V \quad (6.15)$$

Since Eqs. (6.14) and (6.15) are nonlinear, they must be linearized about the operating point in order to cast them in the form of a signal flow graph. Equations (6.14) and (6.15) are linearized by expanding them in a Taylor series about the operating point and retaining only the first-order terms.<sup>†</sup>

$$\begin{aligned} Vd(C_A - \bar{C}_A)/dt = & [-2k\bar{C}_AV - (\bar{Q} + \bar{Q}_A)](C_A - \bar{C}_A) \\ & + \bar{Q}(C_{A_0} - \bar{C}_{A_0}) + (\bar{C}_{A_0} - \bar{C}_A)(Q - \bar{Q}) \\ & + (\rho_A - \bar{C}_A)(Q_A - \bar{Q}_A) \end{aligned} \quad (6.16)$$

$$\begin{aligned} Vd(C_B - \bar{C}_B)/dt = & 2k\bar{C}_AV(C_A - \bar{C}_A) - (\bar{Q} + \bar{Q}_A)(C_B - \bar{C}_B) \\ & - \bar{C}_B(Q - \bar{Q}) - \bar{C}_B(Q_A - \bar{Q}_A) \end{aligned} \quad (6.17)$$

These equations can be handled more easily when written in terms of the

<sup>†</sup> Linearization methods are described in Appendix B.

following deviation variables:

$$\begin{aligned}\theta_Q &= Q - \bar{Q}, & \theta_{A_0} &= C_{A_0} - \bar{C}_{A_0} \\ \theta_{QA} &= Q_A - \bar{Q}_A, & \theta_B &= C_B - \bar{C}_B \\ \theta_A &= C_A - \bar{C}_A\end{aligned}\quad (6.18)$$

The equations describing the reactor are then

$$\begin{aligned}V(d\theta_A/dt) &= [-2k\bar{C}_A V - (\bar{Q} + \bar{Q}_A)]\theta_A \\ &\quad + \bar{Q}\theta_{A_0} + (C_{A_0} - \bar{C}_A)\theta_Q \\ &\quad + (\varrho_A - \bar{C}_A)\theta_{QA},\end{aligned}\quad (6.19)$$

$$\begin{aligned}V(d\theta_B/dt) &= 2k\bar{C}_A V\theta_A - (\bar{Q} + \bar{Q}_A)\theta_B \\ &\quad - \bar{C}_B\theta_Q - \bar{C}_B\theta_{QA}.\end{aligned}\quad (6.20)$$

Taking the Laplace transform of Eqs. (6.19) and (6.20) and rearranging yields

$$\begin{aligned}s\theta_A(s) &= (K_1/T_1)\theta_{A_0}(s) + (K_2/T_1)\theta_{QA}(s) \\ &\quad + (K_3/T_1)\theta_Q(s) - (1/T_1)\theta_A(s)\end{aligned}\quad (6.21)$$

$$\begin{aligned}s\theta_B(s) &= (K_4/T_2)\theta_A(s) - (K_5/T_2)\theta_{QA}(s) \\ &\quad - (K_5/T_2)\theta_Q(s) - (1/T_2)\theta_B(s)\end{aligned}\quad (6.22)$$

where

$$T_1 = V/k_1 \quad (6.23)$$

$$T_2 = V/(\bar{Q} + \bar{Q}_A) \quad (6.24)$$

$$K_1 = \bar{Q}/k_1 \quad (6.25)$$

$$K_2 = (\varrho_A - \bar{C}_A)/k_1 \quad (6.26)$$

$$K_3 = (\bar{C}_{A_0} - \bar{C}_A)/k_1 \quad (6.27)$$

$$K_4 = (2k\bar{C}_A V)/(\bar{Q} + \bar{Q}_A) \quad (6.28)$$

$$K_5 = \bar{C}_B/(\bar{Q} + \bar{Q}_A) \quad (6.29)$$

$$k_1 = 2k\bar{C}_A V + \bar{Q} + \bar{Q}_A \quad (6.30)$$

The error  $E$  which dictates the control action to be taken is computed as

$$E(s) = \theta_{B\text{set}}(s) - \theta_B'(s) \quad (6.31)$$

where  $\theta_B'(s)$  is the delayed signal.  $\theta_B'$  is related to  $\theta_B(s)$  by the transfer function

$$\theta_B'(s)/\theta_B(s) = \exp(-\tau s) \quad (6.32)$$



the constants associated with the proportional, derivative, and integral actions of the controller, respectively. Equation (6.34) can be rearranged to yield

$$M(s) = \frac{K_c T_D}{T_I} \cdot E(s) + \frac{K_c}{T_I} \cdot \frac{1}{s} \cdot E(s) - \frac{1}{T_I} \cdot \frac{1}{s} \cdot M(s) \quad (6.35)$$

The flow rate  $\theta_{QA}(s)$  is related to  $M(s)$  by the equation

$$\theta_{QA}(s) = 1.25M(s) \quad (6.36)$$

where the constant 1.25 represents the gain of the valve which regulates the flow of pure  $A$ .

Equations (6.21), (6.22), (6.31), (6.33), (6.35), and (6.36) completely describe the system. These equations are now cast into the form of a signal flow graph, Fig. 6.9, by defining nodes for the inputs and outputs and connecting them with branches as dictated by the equations. Figure 6.9 is similar to a conventional block diagram in that there are three major sections, or blocks: one for the controller, one for the plant, and one for the transportation lag. As shown in Fig. 6.9, the SFG yields the closed-loop functions for the system. The open-loop functions can be readily obtained

TABLE 6.6  
INVENTORY OF BRANCHES CORRESPONDING TO FIG. 6.9

ROW	OF	P	ORIGIN	END	BRANCH GAIN	POWER OF S
1			1	4	1.00000	0
2			2	10	0.834000E-01	0
3			3	10	-0.734000E-03	0
4			3	12	-0.166670E-02	0
5			4	5	100.000	0
6			5	6	0.666667E-01	0
7			6	7	3.30000	0
8			6	7	1.00000	-1
9			7	8	-1.00000	-1
10			7	9	1.25000	0
11			9	10	0.136600E-01	0
12			9	12	-0.166670E-02	0
13			10	11	1.00000	-1
14			11	12	0.238220	0
15			12	13	1.00000	-1
16			13	14	1.00000	0
17			14	15	22.5000	0
18			14	16	7.05000	0
19			14	17	1.00000	0
20			15	16	-1.00000	-1
21			16	17	-1.00000	-1
22			17	16	7.05000	0
23			17	15	-22.5000	0
24			17	4	-1.00000	0
25			13	12	-0.100000E 00	0
26			11	10	-0.338200	0
27			8	7	0.666667E-01	0



by disconnecting the feedback to the controller; that is, by eliminating the edge from node 17 to node 4 ( $17 \rightarrow 4$ ).

Table 6.6 shows the inventory of branches which correspond to the closed-loop SFG shown in Fig. 6.9. The branch gains were computed using the nominal values of the system parameters. Those branches which are functions of the Laplace operator  $s$  are handled symbolically.

The primary outputs of interest are  $\theta_A$  (node 11) and  $\theta_B$  (node 13); particularly  $\theta_B$  since it is the controlled variable. The transfer functions relating the inputs  $\theta_{Bset}$  (node 1),  $\theta_{A_0}$  (node 2), and  $\theta_Q$  (node 3) to the outputs are generated using the subroutines of Appendix I. Figure 6.10 is a schematic diagram of a program to generate these functions.

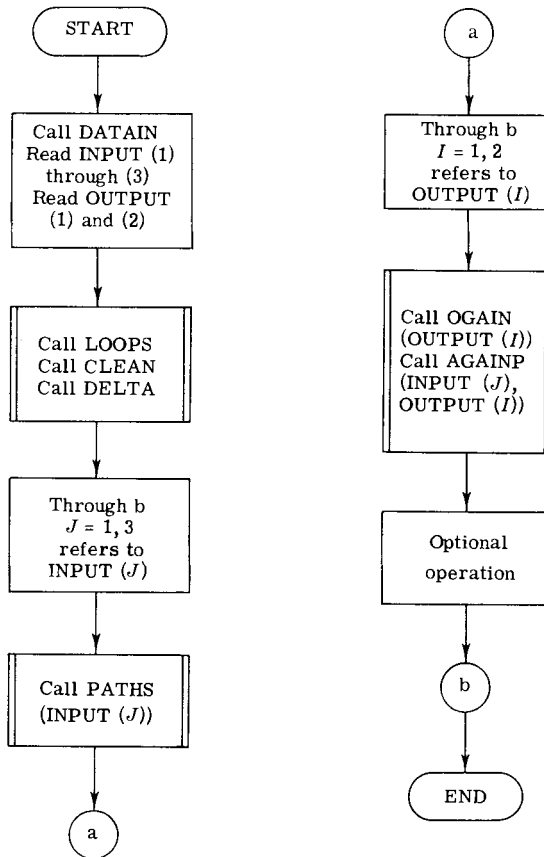


Fig. 6.10. Schematic diagram of a procedure to compute transfer functions for the reactor system, Fig. 6.9.

Note that the last block in Fig. 6.10 refers to an optional operation. After a given transfer function is computed and printed it can be passed along to another routine for further processing such as: generation of values for a root locus, Nyquist, or Bode diagram, determination of poles and/or zeros, inversion to the time domain, or generation of sensitivity functions.

The various transfer functions are obtained as a ratio of two polynomials in  $s$ ,  $N(s)/D(s)$ . Tables 6.7 and 6.8 contain the open-loop transfer functions which relate each input to the outputs  $\theta_B$  and  $\theta_A$ , respectively. The corresponding closed-loop functions are presented in Tables 6.9 and 6.10. Note that the steady-state gain of each transfer function, as computed by the final value theorem, is also presented.

TABLE 6.7

OPEN-LOOP TRANSFER FUNCTIONS RELATING THE INPUTS  $\theta_{Bset}$  (NODE 1),  $\theta_{A0}$  (NODE 2), AND  $\theta_Q$  (NODE 3) TO THE OUTPUT  $\theta_A$  (NODE 11)

FROM NODE 1 TO NODE 11		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.25612468	0.50729968E-01
1	3.4867105	1.4341440
2	9.5308523	11.806135
3	2.7997246	26.122284
4	0.37564951	7.5548649
5	0.0	1.0000000

THE STEADY STATE GAIN IS 0.50487843E 01

FROM NODE 2 TO NODE 11		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.12509983E-01	0.50729968E-01
1	0.31666940	1.4341440
2	1.9750490	11.806135
3	0.60186946	26.122284
4	0.83399951E-01	7.5548649
5	0.0	1.0000000

THE STEADY STATE GAIN IS 0.24659944E 00

FROM NODE 3 TO NODE 11		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	-0.11009989E-03	0.50729968E-01
1	-0.27869954E-02	1.4341440
2	-0.17382331E-01	11.806135
3	-0.52970275E-02	26.122284
4	-0.73399977E-03	7.5548649
5	0.0	1.0000000

THE STEADY STATE GAIN IS -0.21703127E-02

TABLE 6.8

OPEN-LOOP TRANSFER FUNCTIONS RELATING THE INPUTS  $\theta_{Bset}$  (NODE 1),  $\theta_{A_0}$  (NODE 2), AND  $\theta_Q$  (NODE 3) TO THE OUTPUT  $\theta_B$  (NODE 13)

FROM NODE 1 TO NODE 13		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.50445086	0.50729968E-01
1	1.5102396	1.4341440
2	-0.58516490	11.806135
3	-0.26303399	26.122284
4	-0.45834191E-01	7.5548649
5	0.0	1.0000000

THE STEADY STATE GAIN IS 0.99438429E 01

FROM NODE 2 TO NODE 13		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.29801276E-01	0.50729968E-01
1	0.45635718	1.4341440
2	0.14139050	11.806135
3	0.19867532E-01	26.122284
4	0.0	7.5548649
5	0.0	1.0000000

THE STEADY STATE GAIN IS 0.58744913E 00

FROM NODE 3 TO NODE 13		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	-0.11077963E-02	0.50729968E-01
1	-0.19464102E-01	1.4341440
2	-0.43539960E-01	11.806135
3	-0.12599867E-01	26.122284
4	-0.16666998E-02	7.5548649
5	0.0	1.0000000

THE STEADY STATE GAIN IS -0.21837115E-01

Subroutine BODE, which is listed in Appendix I, was used to generate values for both open-loop and closed-loop Bode diagrams. The open-loop and closed-loop values which correspond to the transfer function relating the input  $\theta_{Bset}$  (node 1) to  $\theta_B$  (node 13) are presented in Tables 6.11 and 6.12, respectively. The actual curves are presented in Fig. 6.11. Since the open-loop Bode diagram shows a phase margin of  $30^\circ$  and a gain margin of  $1.0/0.21 = 4.8$ , the system is stable by virtue of the Bode stability criterion. The shape and location of the resonant peak in the closed-loop Bode diagram indicates that the system's time domain transient response to step changes in set point will probably be slow and oscillatory with a long period.

The transfer function relating  $\theta_{Bset}$  to  $\theta_B$ ,  $T_{1 \rightarrow 13}$ , was multiplied by  $1/s$

TABLE 6.9

CLOSED-LOOP TRANSFER FUNCTIONS RELATING THE INPUTS  $\theta_{Bset}$  (NODE 1),  $\theta_{A_0}$  (NODE 2), AND  $\theta_Q$  (NODE 3) TO THE OUTPUT  $\theta_A$  (NODE 11)

FROM NODE 1 TO NODE 11		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.25612468	0.55518079
1	3.4867105	2.6282597
2	9.5308523	10.373603
3	2.7997246	26.505478
4	0.37564951	7.5090303
5	0.0	1.0000000

THE STEADY STATE GAIN IS 0.46133560E 00

FROM NODE 2 TO NODE 11		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	-0.13553008E-01	0.55518079
1	0.23882806	2.6282597
2	2.0008392	10.373603
3	0.59804684	26.505478
4	0.83399951E-01	7.5090303
5	0.0	1.0000000

THE STEADY STATE GAIN IS -0.24411879E-01

FROM NODE 3 TO NODE 11		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.43881051E-02	0.55518079
1	0.10647651E-01	2.6282597
2	-0.21833543E-01	10.373603
3	-0.46372861E-02	26.505478
4	-0.73399977E-03	7.5090303
5	0.0	1.0000000

THE STEADY STATE GAIN IS 0.79039186E-02

and inverted in order to obtain the time domain transient response to changes in set point. Subroutine INVERT was used to accomplish this. Table 6.13 shows the results obtained. Note that the value at  $t = 100$  minutes agrees closely with the steady-state gain of the transfer function presented in Table 6.10. Figure 6.12 shows the transient response of  $\theta_B$  which results from a set point change of 0.1. The same result, as obtained by an analog computer simulation, is also presented in Fig. 6.12. The two curves agree very well.

The techniques of Chapter 5 were used to compute sensitivity functions for the reactor system; the necessary network functions being obtained through use of the subroutines presented in Appendix I. Subroutine

TABLE 6.10

CLOSED-LOOP TRANSFER FUNCTIONS RELATING THE INPUTS  $\theta_{Bset}$  (NODE 1),  $\theta_{A0}$  (NODE 2), AND  $\theta_Q$  (NODE 3) TO THE OUTPUT  $\theta_B$  (NODE 13)

FROM NODE 1 TO NODE 13		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.50445086	0.55518079
1	1.5102396	2.6282597
2	-0.58516490	10.373603
3	-0.26303399	26.505478
4	-0.45834191E-01	7.5090303
5	0.0	1.0000000

THE STEADY STATE GAIN IS 0.90862447E 00

FROM NODE 2 TO NODE 13		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.29801276E-01	0.55518079
1	0.45635718	2.6282597
2	0.14139050	10.373603
3	0.19867532E-01	26.505478
4	0.0	7.5090303
5	0.0	1.0000000

THE STEADY STATE GAIN IS 0.53678505E-01

FROM NODE 3 TO NODE 13		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	-0.11077963E-02	0.55518079
1	-0.19464102E-01	2.6282597
2	-0.43539960E-01	10.373603
3	-0.12599867E-01	26.505478
4	-0.16666998E-02	7.5090303
5	0.0	1.0000000

THE STEADY STATE GAIN IS -0.19953793E-02

SENSE computes the network functions required in order to evaluate sensitivity functions by using Eq. (5.28).

The sensitivity of the transfer function relating  $\theta_{Bset}$  to  $\theta_B$  was determined with respect to the parameter  $1/T_I$ , which appears on branches  $5 \rightarrow 6$  and  $8 \rightarrow 7$  in Fig. 6.9. The sensitivity function with respect to  $T_I$  is simply the negative of that with respect to  $1/T_I$ . The desired sensitivity function is, by Eq. (5.30):

$$\begin{aligned}
 S_{1/T_I}^{T(s, 1/T_I)} = & \left( \frac{D(s, 0)}{D(s, 1/T_I)} - \frac{N(s, 0)}{N(s, 1/T_I)} \right)_{br5 \rightarrow 6} \\
 & + \left( \frac{D(s, 0)}{D(s, 1/T_I)} - \frac{N(s, 0)}{N(s, 1/T_I)} \right)_{br8 \rightarrow 7} \quad (6.37)
 \end{aligned}$$

TABLE 6.11

VALUES FOR AN OPEN-LOOP BODE DIAGRAM CORRESPONDING TO THE  
TRANSFER FUNCTION RELATING  $\theta_{Bset}$  (NODE 1) TO  $\theta_B$  (NODE 13)

OMEGA	GAIN	PHASE ANGLE	ANGLE = 180
0.0	9.9438429	0.0	-180.0000
0.01	9.7862644	-14.3999	-194.3999
0.02	9.3442268	-28.3273	-208.3273
0.03	8.6953964	-41.4070	-221.4070
0.04	7.9327469	-53.4104	-233.4104
0.05	7.1373215	-64.2495	-244.2495
0.06	6.3660011	-73.9402	-253.9402
0.07	5.6517220	-82.5590	-262.5588
0.08	5.0093098	89.7886	-90.2114
0.09	4.4419193	82.9899	-97.0101
0.10	3.9461670	76.9359	-103.0641
0.20	1.4614105	40.7632	-139.2368
0.30	0.73037720	23.5730	-156.4270
0.40	0.43675005	12.6489	-167.3511
0.50	0.29166746	4.5503	-175.4497
0.60	0.20977747	-1.9751	-181.9751
0.70	0.15913510	-7.4941	-187.4941
0.80	0.12566149	-12.3041	-192.3041
0.90	0.10238296	-16.5785	-196.5785
1.00	0.85530281E-01	-20.4263	-200.4263
2.00	0.29316492E-01	-45.3390	-225.3390
3.00	0.17313965E-01	-57.8703	-237.8703
4.00	0.12342177E-01	-65.1359	-245.1359
5.00	0.96256062E-02	-69.7967	-249.7967
6.00	0.79065934E-02	-73.0157	-253.0157
7.00	0.67170449E-02	-75.3631	-255.3631
8.00	0.58430508E-02	-77.1471	-257.1470
9.00	0.51727630E-02	-78.5471	-258.5469
10.00	0.46418868E-02	-79.6741	-259.6741
20.00	0.22990578E-02	-84.8074	-264.8074
30.00	0.15299870E-02	-86.5345	-266.5344
40.00	0.11467747E-02	-87.3999	-267.3999
50.00	0.91715483E-03	-87.9196	-267.9194
60.00	0.76417625E-03	-88.2662	-268.2661
70.00	0.65494608E-03	-88.5137	-268.5137
80.00	0.57304231E-03	-88.6995	-268.6995
90.00	0.50934963E-03	-88.8439	-268.8438

The required network functions are presented in Table 6.14. When these are combined as indicated above, the result is

$$S_{1/T}^{T(s,1/T_1)} = \frac{0.551 + 3.39s + 20.5s^2 + 52.1s^3 + 15.0s^4 + 2.0s^5}{0.55 + 2.63s + 10.34s^2 + 26.5s^3 + 7.51s^4 + 1s^5} - 1.0 \quad (6.38)$$

The real and imaginary parts of this function, for various values of  $s = j\omega$ , yield the corresponding gain and phase sensitivities as indicated by Eq. (5.47).

TABLE 6.12

VALUES FOR A CLOSED-LOOP BODE DIAGRAM CORRESPONDING TO THE  
TRANSFER FUNCTION RELATING  $\theta_{Bset}$  (NODE 1) TO  $\theta_B$  (NODE 13)

OMEGA	GAIN	PHASE ANGLE	ANGLE - 180
0.0	0.90862447	0.0	-180.0000
0.01	0.90981793	-0.9981	-180.9981
0.02	0.91341519	-2.0021	-182.0021
0.03	0.91946661	-3.0182	-183.0182
0.04	0.92805588	-4.0526	-184.0526
0.05	0.93930733	-5.1123	-185.1123
0.06	0.95338619	-6.2045	-186.2045
0.07	0.97050595	-7.3373	-187.3373
0.08	0.99093866	-8.5201	-188.5201
0.09	1.0150185	-9.7633	-189.7633
0.10	1.0431623	-11.0795	-191.0795
0.20	1.7459173	-34.1732	-214.1732
0.30	2.2128391	52.9055	-127.0945
0.40	0.77491844	11.3007	-168.6993
0.50	0.40535057	-0.8615	-180.8615
0.60	0.25830364	-7.9126	-187.9126
0.70	0.18281698	-13.1729	-193.1729
0.80	0.13819802	-17.5310	-197.5310
0.90	0.10934871	-21.3259	-201.3259
1.00	0.89488864E-01	-24.7175	-204.7175
2.00	0.28884403E-01	-46.7729	-226.7729
3.00	0.17021567E-01	-57.9931	-237.9931
4.00	0.12224853E-01	-64.6934	-244.6934
5.00	0.96058249E-02	-69.2586	-249.2586
6.00	0.79261661E-02	-72.5848	-252.5848
7.00	0.67471825E-02	-75.0751	-255.0751
8.00	0.58726892E-02	-76.9781	-256.9780
9.00	0.51985942E-02	-78.4645	-258.4644
10.00	0.46634562E-02	-79.6512	-259.6511
20.00	0.23028729E-02	-84.8988	-264.8987
30.00	0.15311772E-02	-86.6101	-266.6099
40.00	0.11472846E-02	-87.4605	-267.4604
50.00	0.91741933E-03	-87.9695	-267.9692
60.00	0.76432899E-03	-88.3084	-268.3083
70.00	0.65504294E-03	-88.5503	-268.5503
80.00	0.57310704E-03	-88.7317	-268.7314
90.00	0.50939526E-03	-88.8727	-268.8726

Network functions similar to those of Table 6.14 are presented in Table 6.15. These functions, however, correspond to the controller gain  $K_c$  which appears on branch 4  $\rightarrow$  5 in Fig. 6.9. Since the parameter only appears on one branch, and the leakage transmission is zero, the desired sensitivity function is

$$S_{K_c}^{T(s, K_c)} = 1/R_{K_c}(s, K_c) = D(s, 0)/D(s, K_c) \quad (6.39)$$

$$S_{K_c}^{T(s, K_c)} = \frac{0.0507 + 1.43s + 11.8s^2 + 26.1s^3 + 7.55s^4 + s^5}{0.555 + 2.63s + 10.4s^2 + 26.5s^3 + 7.51s^4 + s^5} \quad (6.40)$$

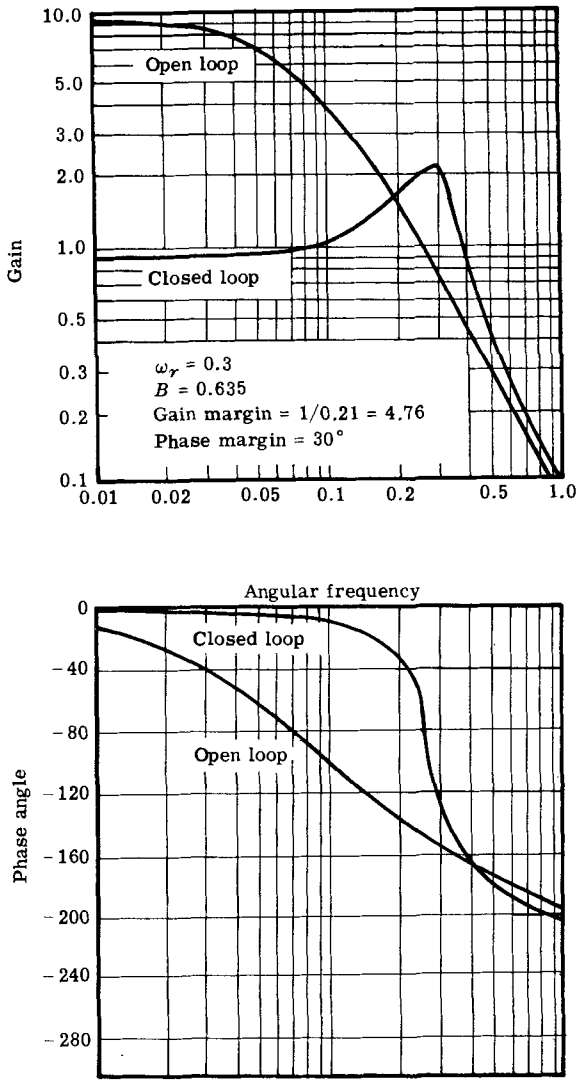
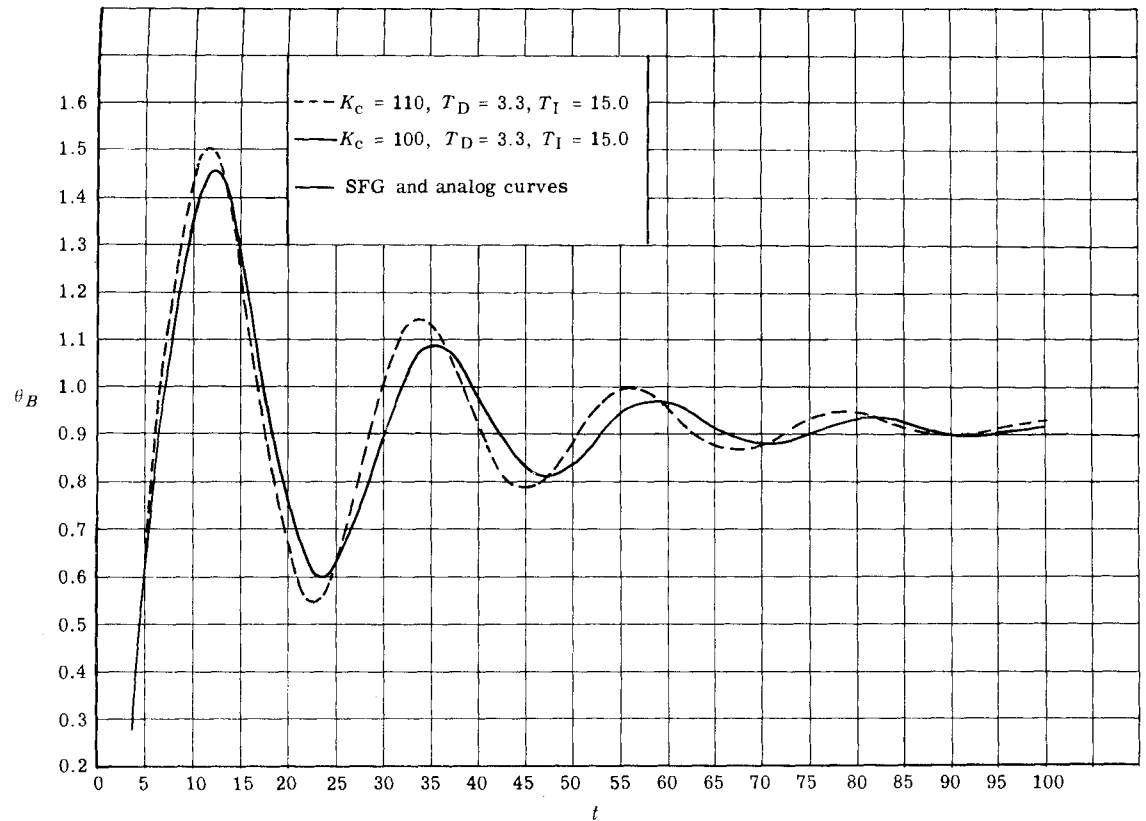


Fig. 6.11. Open-loop and closed-loop Bode diagrams for the reactor control system.



TABLE 6.13  
VALUES OF  $\theta_B(t)$  RESULTING FROM A UNIT STEP CHANGE  
IN  $\theta_{Bset}$

TIME	QUIPUI	FIRST DERIV.
0.0	0.17881393E-06	-0.45834240E-01
2.000	0.54691136E-01	0.92175126E-01
4.000	0.33455843	0.17768866
6.000	0.72292638	0.20006752
8.000	1.0959606	0.16463363
10.000	1.3558540	0.91014206E-01
12.000	1.4521589	0.55527464E-02
14.000	1.3863705	-0.66823244E-01
16.000	1.2052603	-0.10898823
18.000	0.97547376	-0.11483878
20.000	0.76718861	-0.89026809E-01
22.000	0.63233167	-0.43868512E-01
24.000	0.59432417	0.52617379E-02
26.000	0.64670932	0.44562593E-01
28.000	0.75993761	0.65222919E-01
30.000	0.89348906	0.64920902E-01
32.000	1.0080652	0.47330748E-01
34.000	1.0763664	0.20114463E-01
36.000	1.0880842	-0.78086890E-02
38.000	1.0497866	-0.28859500E-01
40.000	0.98027778	-0.38593672E-01
42.000	0.90347219	-0.36351327E-01
44.000	0.84113503	-0.24784390E-01
46.000	0.80743510	-0.85807592E-02
48.000	0.80646420	0.71369223E-02
50.000	0.83287078	0.18253975E-01
52.000	0.87490892	0.22589862E-01
54.000	0.91867095	0.20148955E-01
56.000	0.95217317	0.12744967E-01
58.000	0.96822619	0.32061734E-02
60.000	0.96553391	-0.55535734E-02
62.000	0.94802868	-0.11328608E-01
64.000	0.92292506	-0.13047876E-01
66.000	0.89823121	-0.11050001E-01
68.000	0.88047105	-0.64101480E-02
70.000	0.87320024	-0.85360883E-03
72.000	0.87656236	0.39783642E-02
74.000	0.88783646	0.69193467E-02
76.000	0.90265536	0.75081475E-02
78.000	0.91645324	0.59906468E-02
80.000	0.92571980	0.31332152E-02
82.000	0.92875677	-0.71151298E-04
84.000	0.92582107	-0.27072877E-02
86.000	0.91872048	-0.41684322E-02
88.000	0.91006577	-0.42657740E-02
90.000	0.90243441	-0.32071183E-02
92.000	0.89768940	-0.14733644E-02
94.000	0.89660329	0.35668677E-03
96.000	0.89883888	0.17776999E-02
98.000	0.90322953	0.24803302E-02
100.000	0.90823328	0.24000246E-02



**Fig. 6.12.** Transient response of  $\theta_B$  to a step change of 0.1 in  $\theta_{Bset}$ , three cases. ---  $K_c = 110, T_D = 3.3, T_I = 15.0$ ; —  $K_c = 100, T_D = 3.3, T_I = 15.0$ ; — SFG and analog curves.

TABLE 6.14  
NETWORK FUNCTIONS REQUIRED TO COMPUTE  $S_{T_1}^{(s,1/T_1)}$

FROM NODE 1 TO NODE 13		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.50445086	0.55518079
1	1.5102396	2.6282597
2	-0.58516490	10.373603
3	-0.26303399	26.505478
4	-0.45834191E-01	7.5090303
5	0.0	1.0000000

$$T(s, 1/T_1)$$

FROM NODE 1 TO NODE 13		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.0	0.50729968E-01
1	0.0	1.4341440
2	0.0	11.806135
3	0.0	26.122284
4	0.0	7.5548649
5	0.0	1.0000000

$$T(s, 0)_{br\ 5+6}$$

FROM NODE 1 TO NODE 13		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.50445086	0.50445086
1	1.5102396	1.9550657
2	-0.58516490	8.6653957
3	-0.26303399	26.006302
4	-0.45834191E-01	7.4423637
5	0.0	1.0000000

$$T(s, 0)_{br\ 8+7}$$

The effect of variations in the value of  $K_c$  on the step response of the system is determined most easily by evaluating the transfer function again and inverting, or by determining the shift in the poles and zeros of the transfer function and inverting. Equation (6.40), which is the reciprocal of return difference, can be used in Eqs. (5.48) and (5.49) to evaluate the shift in the poles of the transfer function which correspond to a given change in  $K_c$ . Since the leakage transmission is zero, the null-return difference is zero and, therefore, Eqs. (6.34) and (6.35) predict no shifts in the zeros of the transfer function. Table 6.16 shows the poles of the original transfer function ( $K_c = 100$ ) and the shift caused by a  $\Delta K_c$  of 10 ( $K_c = 110$ ), as well as the new poles of the transfer function. The resulting transient response, as obtained by subroutine INVERT, is plotted along with the other response curves in Fig. 6.12.

TABLE 6.15  
NETWORK FUNCTIONS REQUIRED TO COMPUTE  $S_{K_c}^{T(s, K_c)}$

FROM NODE 1 TO NODE 13		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.50445086	0.55518079
1	1.5102396	2.6282597
2	-0.58516490	10.373603
3	-0.26303399	26.505478
4	-0.45834191E-01	7.5090303
5	0.0	1.0000000
$T(s, K_c)$		
FROM NODE 1 TO NODE 13		
POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	0.0	0.50729968E-01
1	0.0	1.4341440
2	0.0	11.506135
3	0.0	26.122284
4	0.0	7.5548649
5	0.0	1.0000000
$T(s, 0)$		

TABLE 6.16

POLES OF THE TRANSFER FUNCTION RELATING  $\theta_{B\text{set}}$  TO  $\theta_B$  AND THE SHIFTS CAUSED BY A  $\Delta K_c = 10$ . TOP: POLES OF  $T(s)_{1 \rightarrow 13}$ . MIDDLE: SHIFT IN POLES CAUSED BY  $\Delta K_c = 10$ . BOTTOM: NEW POLES OF  $T(s)_{1 \rightarrow 13}$

N	REAL	IMAGINARY
6	-0.59604645E-07	0.0
5	-0.31466895	0.0
4	-0.47012508E-01	0.26999456
3	-0.47012508E-01	-0.26999456
2	-3.5501041	3.3007259
1	-3.5501041	-3.3007259
N	Real	Imag
1	0.0	0.0
2	+0.00077396	0.0
3	+0.04100442E-01	+0.01273662
4	+0.04100442E-01	-0.01273662
5	-0.00224120	+0.01224430
6	-0.00224120	-0.01224430
N	REAL	IMAGINARY
6	0.0	0.0
5	-0.31389499	0.0
4	-0.42912066E-01	0.28273118
3	-0.42912066E-01	-0.28273118
2	-3.5523453	3.3129702
1	-3.5523453	-3.3129702

### Eigenvalue Problems

Equation (6.41) represents, in matrix notation, a system of  $n$  equations in  $n$  unknowns, where  $\mathbf{A}$  is the matrix of coefficients,  $\mathbf{X}$  is the vector of unknowns, and  $\mathbf{B}$  is the vector of inputs, or right-hand sides.

$$\mathbf{AX} = \mathbf{B} \quad (6.41)$$

During the course of analyzing physical systems, engineers often encounter the problem of determining the *eigenvalues*  $\lambda$  and the *eigenvectors*  $\mathbf{X}$  which satisfy the system of equations

$$\mathbf{AX} = \lambda \mathbf{X} \quad (6.41)$$

or

$$(\mathbf{A} - \lambda \mathbf{I})\mathbf{X} = 0 \quad (6.42)$$

where  $\mathbf{I}$  is the identity matrix. Nontrivial solutions of this problem exist only when the *characteristic determinant* is equal to zero.

$$\det(\mathbf{A} - \lambda \mathbf{I}) = \begin{vmatrix} a_{11} - \lambda & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} - \lambda & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} - \lambda \end{vmatrix} = 0 \quad (6.43)$$

Expanding the determinant in Eq. (6.43) yields the characteristic polynomial for the system of equations. The  $n$  roots of this  $n$ th degree polynomial are the desired eigenvalues. The eigenvector which corresponds to the  $i$ th eigenvalue is given by one of the nonzero columns of the adjoint matrix,  $\text{adj}(\mathbf{A} - \lambda_i \mathbf{I})$ .

In searching the literature one finds many routines for computing the real eigenvalues of matrices which are symmetric about the main diagonal, and numerical methods of obtaining the greatest real eigenvalue. Routines which yield all of the real and complex eigenvalues of a nonsymmetric matrix are much less prominent.

A major difficulty lies in expanding the characteristic determinant to obtain the characteristic equation. Once the characteristic equation is obtained, the eigenvalues can be found by using a generalized root finding technique such as Barstow's method (see subroutine ROOTS, Appendix I).

The system of equations represented by Eq. (6.42) can be cast into the form of a SFG. If the  $j$ th equation is solved for the  $j$ th unknown, all of the branch gains will be of the form  $a\lambda^k$ , where  $k$  is either 0 or 1 and  $a$  is some

constant. The characteristic determinant,  $\det(\mathbf{A} - \lambda\mathbf{I})$ , can then be evaluated using subroutine DELTA and the result, when equated to zero, will be the characteristic polynomial for the system. Subsequently, subroutine ROOTS can be used to obtain the roots of the characteristic polynomial; these are the desired eigenvalues.

As a specific example, the following system of equations is considered:

$$\begin{bmatrix} -54.82 - \lambda & 1.06 & 0.1 & 25.63 \\ 3.17 & -20.87 - \lambda & 14.82 & 0.07 \\ 0.39 & 19.77 & -16.82 - \lambda & 3.88 \\ 51.26 & 0.04 & 1.94 & -29.58 - \lambda \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = 0 \quad (6.44)$$

The  $i$ th of these equations can be solved for the  $i$ th unknown to yield

$$\begin{aligned} x_1 &= -0.01824\lambda x_1 + 0.01933x_2 + 0.001824x_3 + 0.46753x_4 \\ x_2 &= 0.1519x_1 - 0.0479\lambda x_2 + 0.7101x_3 + 0.00335x_4 \\ x_3 &= 0.02318x_1 + 1.1754x_2 - 0.05945\lambda x_3 + 0.2307x_4 \\ x_4 &= 1.733x_1 + 0.00135x_2 + 0.06558x_3 - 0.0338\lambda x_4 \end{aligned} \quad (6.45)$$

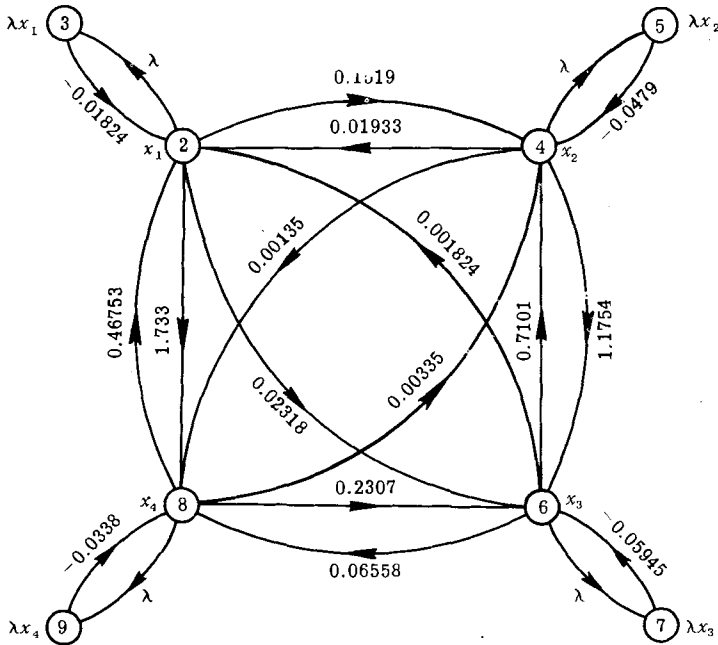


Fig. 6.13. SFG formulation for eigenvalue determination, Eq. (6.45).

TABLE 6.17

INVENTORY OF BRANCHES CORRESPONDING TO FIG. 6.13 (TOP),  
CHARACTERISTIC DETERMINANT (MIDDLE), AND EIGENVALUES FOR  
EQ. (6.44) (BOTTOM)

ROW	WE	P	ORIGIN	END	BRANCH GAIN	POWER OF S
1			2	3	1.00000	1
2			2	4	0.151900	0
3			2	6	0.231800E-01	0
4			2	8	1.73300	0
5			4	5	1.00000	1
6			4	6	1.17540	0
7			4	8	0.135000E-02	0
8			6	7	1.00000	1
9			6	8	0.655800E-01	0
10			8	9	1.00000	1
11			9	8	-0.338000E-01	0
12			8	6	0.230700	0
13			8	4	0.335000E-02	0
14			8	2	0.467530	0
15			7	6	-0.594500E-01	0
16			6	4	0.710100	0
17			6	2	0.182400E-02	0
18			5	4	-0.479000E-01	0
19			4	2	0.193300E-01	0
20			3	2	-0.182400E-01	0

POWER OF S	NUMERATOR COEFF.	DENOMINATOR COEFF.
0	-0.44606207E-03	1.0000000
1	0.27586382E-01	0.0
2	0.62096193E-02	0.0
3	0.21437446E-03	0.0
4	0.17556122E-05	0.0

N	REAL	IMAGINARY
4	0.16110420E-01	0.0
3	-5.4275370	0.0
2	-36.011734	0.0
1	-80.684921	0.0

The SFG representation of Eqs. (6.45) is given in Fig. 6.13. The corresponding set of branches is shown in Table 6.16. In this case the symbol  $s$  corresponds to  $\lambda$ . Obtaining the characteristic determinant required the calling of subroutines DATAIN, LOOPS, DELTA, and DELTAP; in that order. The characteristic determinant computed is shown in Table 6.17. Also shown are the roots of the characteristic polynomial (the eigenvalues) which were obtained through the use of subroutine ROOTS.

### Ordering Recycle Calculations

When simulating chemical processes containing recycle loops, it is common practice to assume values of temperature, pressure, and other unknown quantities associated with one or more of the recycle streams in

order to initiate the process calculations. The streams for which values are assumed are said to have been *torn*. The problem of determining how to choose the *best* tear(s) has been the subject of much discussion [3–9] and a number of criteria for choosing the best tear(s) have been proposed. The simplest criterion, and that most commonly used (though not necessarily the correct criteria) is that the streams to be torn should be chosen such that the total number of tears is a minimum.

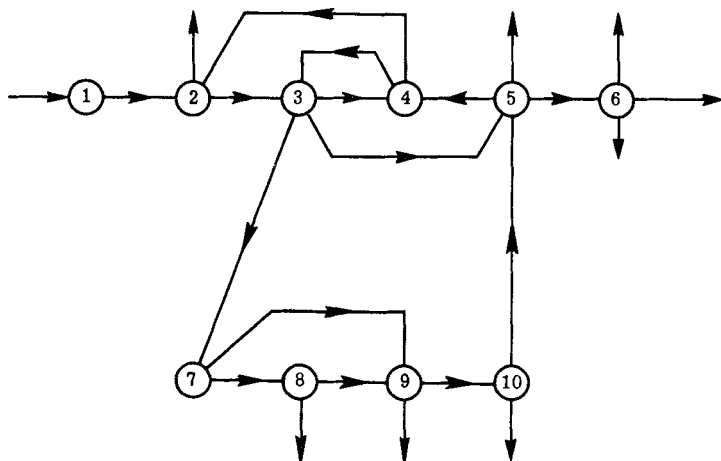


Fig. 6.14. Flow-graph representation of a hypothetical chemical process.

Figure 6.14 is a flow-graph representation of a process flow sheet corresponding to a hypothetical plant. The nodes represent the various pieces of equipment and the edges represent streams. The inventory of branches corresponding to Fig. 6.14 and the recycle loops, as obtained by subroutine LOOPS, are presented in Table 6.18. All branch and loop gains are zero since they are of no concern in this problem. Each recycle loop can be conveniently referenced by the “row of  $P$ ” number which appears in Table 6.18.

Once the loops are enumerated, they can be used to generate the *cycle matrix* shown in Table 6.19. The cycle matrix clearly indicates which streams appear in each recycle loop. The *stream frequency* corresponding to each stream shows how many loops contain that stream. In determining which streams to tear, it is best to tear those streams which have the highest stream frequencies, if the total number of tears is to be a minimum. It is evident from the cycle matrix (Table 6.19) that if stream  $5 \rightarrow 4$  is torn, six of the eight recycle loops will be broken (43, 44, 45, 47, 48, 49). The two



TABLE 6.18  
INVENTORY OF BRANCHES (TOP) AND RECYCLE LOOPS (BOTTOM)  
CORRESPONDING TO FIG. 6.14

ROW OF P	ORIGIN	END	BRANCH GAIN	POWER OF S
1	1	2	0.0	0
2	2	3	0.0	0
3	3	4	0.0	0
4	3	5	0.0	0
5	3	7	0.0	0
6	5	6	0.0	0
7	7	8	0.0	0
8	7	9	0.0	0
9	8	9	0.0	0
10	9	10	0.0	0
11	10	5	0.0	0
12	5	4	0.0	0
13	4	3	0.0	0
14	4	2	0.0	0

ROW OF P	TRACE	GAIN	POWER OF S
42	3--> 4--> 3-->	0.0	0
43	3--> 5--> 4--> 3-->	0.0	0
44	3--> 7--> 9--> 10-->	0.0	0
	5--> 4--> 3-->	0.0	0
45	3--> 7--> 8--> 9-->	0.0	0
	10--> 5--> 4--> 3-->	0.0	0
46	2--> 3--> 4--> 2-->	0.0	0
47	2--> 3--> 5--> 4-->	0.0	0
	2-->	0.0	0
48	2--> 3--> 7--> 9-->	0.0	0
	10--> 5--> 4--> 2-->	0.0	0
49	2--> 3--> 7--> 8-->	0.0	0
	9--> 10--> 5--> 4-->	0.0	0
	2-->	0.0	0

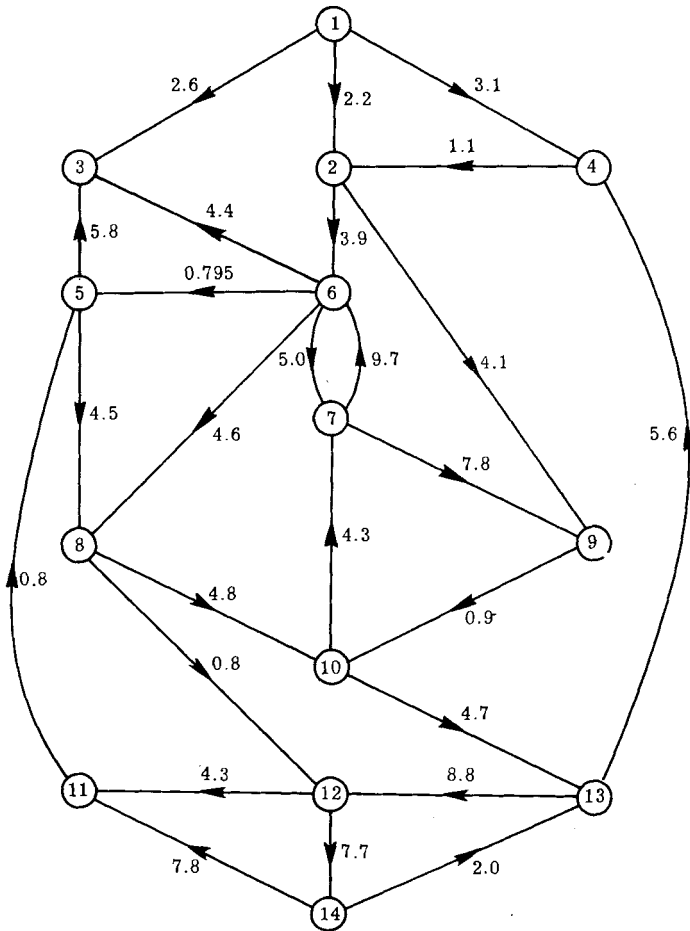
TABLE 6.19  
CYCLE MATRIX FOR THE CHEMICAL PROCESS SHOWN IN FIG. 6.14

Stream		2→3	3→4	3→5	3→7	7→8	7→8	8→9	9→10	10→5	5→4	4→3	4→2
Loop													
42	①											1	
43		1								①	1		
44				1		1		1	1	①	1		
45				1	1			1	1	1	①	1	
46	1	①											1
47	1		1							①		1	
48	1			1		1		1	1	①		1	
49	1			1	1			1	1	1	①		1
Stream frequency		4	2	2	4	2	2	2	4	4	6	4	4

unbroken loops (46, 42) can be broken by tearing stream  $3 \rightarrow 4$ . The minimum number of tears is, therefore, two ( $5 \rightarrow 4$ ,  $3 \rightarrow 4$ ) and the recycle computations can be carried out in the following order: 4, 2, 3, 7, 8, 9, 10, 5.

**Transportation Problems**

The solution of transportation problems by flow-graph methods generally requires that a path or loop be found which meets certain specifications with regard to length or number of steps. Subroutines PATHS and LOOPS



**Fig. 6.15.** Flow-graph representation of a transportation problem.

TABLE 6.20  
INVENTORY OF BRANCHES CORRESPONDING TO FIG. 6.15

ROW OF P	ORIGIN	END	BRANCH GAIN	POWER OF S
1	1	2	2.20000	1
2	1	3	2.60000	1
3	1	4	3.10000	1
4	2	6	3.90000	1
5	2	9	4.10000	1
6	5	8	4.50000	1
7	5	7	5.00000	1
8	6	8	4.60000	1
9	7	9	7.80000	1
10	8	10	4.80000	1
11	8	12	0.300000	1
12	9	10	0.900000	1
13	10	13	4.70000	1
14	12	14	7.70000	1
15	14	13	2.00000	1
16	14	11	7.80000	1
17	13	12	6.80000	1
18	13	4	5.60000	1
19	12	11	4.30000	1
20	11	5	0.800000	1
21	10	7	4.30000	1
22	7	6	9.70000	1
23	5	5	0.795000	1
24	6	3	4.40000	1
25	5	3	5.80000	1
26	4	2	1.10000	1

lend themselves quite well to the solution of transportation problems since they efficiently enumerate all of the paths and/or loops in a given flow graph. The numerical value associated with each edge (time, distance, etc.) can be input just as though they were branch gains in a signal flow graph. Changing the multiplication sign in statement 85 of subroutine LOOPS to a plus sign causes the loop and path gains to be computed as a sum of branch gains rather than a product.

Since the "power of  $s$ " associated with a given path or loop is the sum of the "powers of  $s$ " corresponding to its constituent branches, the "power of  $s$ " capability can be used to flag specific branches or to indicate the number of branches which compose a given path or loop.

As a specific example, consider the flow graph shown in Fig. 6.15. The corresponding inventory of branches is given in Table 6.20. Note that the gain specified for each branch is equal to its length, and that the "power of  $s$ " specified for each branch is unity.

The paths from node 1 to node 14 obtained by using subroutine PATHS, are presented in Table 6.21. The "row of  $P$ " number specified for each path serves as a convenient reference number. The gains given for the various paths indicate their lengths, and the "power of  $s$ " given for each path tells

TABLE 6.21

PATHS IN THE GRAPH OF FIG. 6.15 WHICH LEAD FROM NODE 1 TO NODE 14

ROW	OF P	1	ISACE	-->	-->	-->	-->	GAIN	POWER OF 5
42		1-->	2-->	6-->	8-->			19.2000	5
99		12-->	14-->						
		1-->	2-->	6-->	8-->				
100		10-->	13-->	12-->	14-->			36.7000	7
		1-->	2-->	9-->	10-->				
		13-->	12-->	14-->				28.4000	6
101		1-->	2-->	6-->	7-->				
		9-->	10-->	13-->	12-->				
		14-->						41.0000	8
102		1-->	2-->	6-->	5-->				
		8-->	12-->	14-->				19.8950	6
103		1-->	2-->	9-->	10-->				
		7-->	6-->	5-->	8-->				
		12-->	14-->					34.9950	9
104		1-->	2-->	9-->	10-->				
		7-->	6-->	8-->	12-->				
		14-->						34.3000	8
105		1-->	4-->	2-->	6-->				
		8-->	12-->	14-->				21.2000	6
177		1-->	2-->	6-->	5-->				
		8-->	10-->	13-->	12-->				
		14-->						37.3950	8
178		1-->	4-->	2-->	6-->				
		8-->	10-->	13-->	12-->				
		14-->						38.7000	3
179		1-->	4-->	2-->	9-->				
		10-->	13-->	12-->	14-->			30.4000	7
180		1-->	4-->	2-->	6-->				
		7-->	9-->	10-->	13-->				
		12-->	14-->					43.0000	9
181		1-->	4-->	2-->	6-->				
		5-->	8-->	12-->	14-->			21.8950	7
182		1-->	4-->	2-->	9-->				
		10-->	7-->	6-->	5-->				
		8-->	12-->	14-->				36.9950	10
183		1-->	4-->	2-->	9-->				
		10-->	7-->	6-->	8-->				
		12-->	14-->					36.3000	9
213		1-->	4-->	2-->	6-->				
		5-->	8-->	10-->	13-->				
		12-->	14-->					32.3950	9

how many steps (edges) are in the path. In terms of both distance and number of steps, path 42 (length = 19.2, 5 steps) is the shortest path from node 1 to node 14. Similarly, the longest paths from node 1 to node 14, in terms of length and maximum number of steps, are path 180 (length = 43.0, 9 steps) and path 182 (length = 36.995, 10 steps), respectively.

## REFERENCES

1. T. Takamatsu, I. Hashimoto, and H. Ohno, Optimal design of a large complex system from the viewpoint of sensitivity analysis. *Ind. Eng. Chem. Process Des. Develop.* **9**, 368 (1970).

2. D. R. Coughanowr and L. B. Koppel, "Process Systems Analysis and Control." McGraw-Hill, New York, 1965.
3. J. H. Christensen and D. F. Rudd, *AIChE J.* **15**, 94 (1969).
4. W. Lee and D. F. Rudd, On the ordering of recycle calculations. *AIChE J.* **12**, 1184 (1966).
5. R. W. H. Sargent and A. W. Westerberg, *Trans. Inst. Chem. Eng.* **42**, 190 (1964).
6. W. P. Ledet and D. M. Himmelblau, *Advan. Chem. Eng.* **8**, 185 (1970).
7. R. L. Norman, A matrix method of location of cycles of a directed graph. *AIChE J.* **11**, 450 (1965).
8. R. S. Upadhye and E. A. Grens, *AIChE J.* **18**, 53 (1972).
9. H. Weinblatt, *J. Assoc. Comput. Mach.* **19**, 43 (1972).

## CHAPTER 7

# LINEAR AND NONLINEAR PROGRAMMING<sup>†</sup>

This chapter introduces the application of flow graphs to optimization problems. First, a linear programming (LP) algorithm in which SFG methods rather than matrix manipulations are used is developed and applied. Next we consider nonlinear optimization problems and develop optimization procedures based on gradient methods and the newly developed LP algorithm [1]. This technique is a variation of Zoutendijk's method of feasible directions. Those readers wishing a review of linear programming, and one-dimensional (Fibonacci) search techniques are referred to Appendixes E and G.

### General Form of LP Problems

Linear programming problems are stated as:

Maximize:

$$Z = C_1x_1 + C_2x_2 + \cdots + C_nx_n \quad (7.1)$$

<sup>†</sup> Principal author: Takashi Tonomura, Cullen College of Engineering, University of Houston, Houston, Texas (on leave from Toray Industries, Inc., Tokyo, Japan).



TABLE 7.1  
EXAMPLE FOR SIMPLEX ALGORITHM

Problem: Maximize

$$Z = x_1 + 2x_2,$$

Subject to

$$\begin{aligned} x_1 + x_2 &\leq 10, \\ 5x_1 + x_2 &\leq 40, \\ x_1 + 5x_2 &\leq 40, \quad x_j \geq 0. \end{aligned}$$

Reformulation of problem:

$$\begin{aligned} Z - x_1 - 2x_2 &= 0 \\ x_1 + x_2 + x_3 &= 10 \\ 5x_1 + x_2 + x_4 &= 40 \\ x_1 + 5x_2 + x_5 &= 40, \quad x_j \geq 0 \end{aligned}$$

Simplex tableau<sup>a</sup>:

	<i>Z</i>	<i>x</i> <sub>1</sub>	<i>x</i> <sub>2</sub>	<i>x</i> <sub>3</sub>	<i>x</i> <sub>4</sub>	<i>x</i> <sub>5</sub>	<i>b</i>	
Step 1	1	−1	−2	0	0	0	0	<i>x</i> <sub>2</sub> enters
	0	1	1	1	0	0	10	
	0	5	1	0	1	0	40	<i>x</i> <sub>5</sub> leaves
	0	1	5	0	0	1	40	
Step 2	1	− $\frac{3}{5}$	0	0	0	$\frac{2}{5}$	16	<i>x</i> <sub>1</sub> enters
	0	$\frac{4}{5}$	0	1	0	− $\frac{1}{5}$	2	
	0	$\frac{24}{5}$	0	0	1	− $\frac{1}{5}$	32	<i>x</i> <sub>3</sub> leaves
	0	$\frac{1}{5}$	1	0	0	$\frac{1}{5}$	8	
Step 3	1	0	0	$\frac{3}{4}$	0	$\frac{1}{4}$	17.5	Optimal
	0	1	0	$\frac{5}{4}$	0	− $\frac{1}{4}$	2.5	
	0	0	0	−6	1	1	20.0	
	0	0	1	− $\frac{1}{4}$	0	$\frac{1}{4}$	7.5	

Optimum solution:

$$\begin{aligned} Z &= 17.5, & x_1 &= 2.5, & x_2 &= 7.5, \\ x_3 &= 0.0, & x_4 &= 20.0, & x_5 &= 0.0. \end{aligned}$$

<sup>a</sup> Only the coefficients are shown.



$$\begin{array}{rcl}
(0) & Z - C_1'x_1 - C_2'x_2 - \cdots - C_N'x_N & = b_0' \\
(1) & a_{11}'x_1 + a_{12}'x_2 + \cdots + a_{1N}'x_N & = b_1' \\
& \vdots & \\
& \vdots & \\
(m) & a_{m1}'x_1 + a_{m2}'x_2 + \cdots + a_{mN}'x_N & = b_m'
\end{array} \tag{7.5}$$

where  $N = m + n$ .

Only one basic variable with a coefficient of unity exists in each of relations [(1) – (m)] of Eq. (7.5), so these equations can be arranged in the diagonal form of Eqs. (7.4).

7. Repeat steps 3 to 6 until the optimum solution is obtained.

An example is shown in the form of a simplex tableau in Table 7.1.

### Simplex Solution by Signal-Flow-Graph Methods

In step 6, we set all the nonbasic variables equal to zero and solved the equations obtained. This is done by Gauss–Jordan elimination in the simplex algorithm. It can also be done to advantage by SFG analysis.

The simplex algorithm tells us to choose the nonbasic variable ( $x_i$ ) which has the smallest (i.e., negative largest) coefficient in relation (0) of Eq. (7.5) as an entering basic variable, and the basic variable ( $x_j$ ) in relation (1) which has the smallest positive  $b_i'/a_{ij}'$  as a leaving basic variable. Now, suppose that  $x_{m+1}, \dots, x_{n+m-1}$  and  $x_{n+m}$  are the nonbasic variables, and  $x_1, \dots, x_{m-1}$ , and  $x_m$  are the basic variables at a certain step. Then Eq. (7.5) may be rewritten as

$$\begin{aligned}
Z &= C_{m+1}'x_{m+1} + \cdots + C_{n+m}'x_{n+m} + b_0' \\
x_i &= -a_{j,m+1}'x_{m+1} - \cdots - a_{j,n+m}'x_{n+m} + b_j'
\end{aligned} \tag{7.6}$$

where  $i = 1, \dots, m, j = 1, \dots, m$ .

This implies that the basic variables and  $Z$  can be obtained in terms of the nonbasic variables using relation (1)–(m) in Eq. (7.4). Therefore:

1.  $C_M'$  is the contribution of  $x_M$  to  $Z$ ; that is, the transmittance from  $x_M$  to  $Z$  satisfying constraint relation (1)–(m) in Eq. (7.5);

2.  $-a_{jN}'$  is the contribution of  $x_N$  to  $x_i$ ; that is, the transmittance from  $x_N$  to  $x_i$  satisfying the set of relations (1)–(m) in Eq. (7.5).

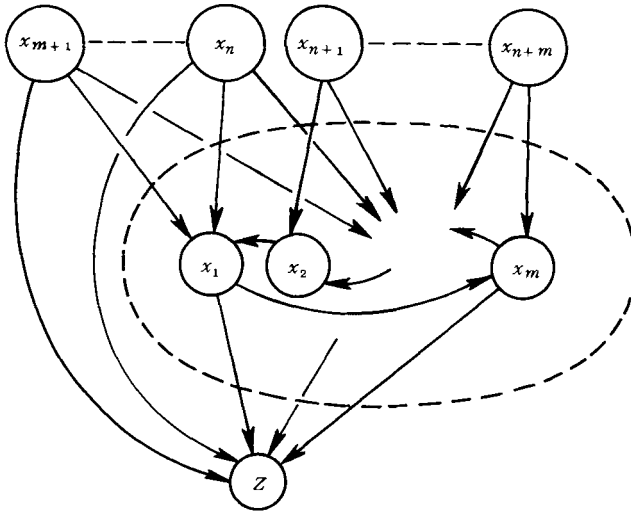


Fig. 7.1. SFG representation. ( $m < n$  is assumed here so that at least  $(n - m)$  input nodes which are nonbasic variables have direct connection to  $Z$  in addition to many branches to the other nodes.)

Thus, once the basic variables and nonbasic variables are specified,  $C_M'$  and  $-a'_{jN}$  are calculated as the appropriate transmittances by Mason's routine using the SFG shown in Fig. 7.1 for the case of Eq. (7.6).

### Example Problems

The same problem that was solved by use of simplex tableaux in Table 7.1 is now solved by the SFG method. A stepwise procedure is followed: each step corresponds to the step with the same number in Table 7.1.

Problem

Maximize

$$Z = x_1 + 2x_2 \quad (0)$$

Subject to

$$x_1 + x_2 \leq 10 \quad (1)$$

$$5x_1 + x_2 \leq 40 \quad (2)$$

$$x_1 + 5x_2 \leq 40 \quad (3)$$

and

$$x_1, x_2 \geq 0$$

*Step 1.* Rewrite the equations, introducing the slack variables  $x_3, x_4, x_5$ :

$$\begin{aligned} (0) \quad Z &= x_1 + 2x_2 \\ (1) \quad x_1 + x_2 + x_3 &= 10 \\ (2) \quad 5x_1 + x_2 + x_4 &= 40 \\ (3) \quad x_1 + 5x_2 + x_5 &= 40 \end{aligned}$$

Assign  $x_3(1)$ ,  $x_4(2)$ , and  $x_5(3)$  as the basic variables, where the numbers in parenthesis are the numbers of the equation to which they are assigned.

Rewrite the equations

$$\begin{aligned} Z &= x_1 + 2x_2 \\ x_3 &= 10 - x_1 - x_2 \\ x_4 &= 40 - 5x_1 - x_2 \\ x_5 &= 40 - x_1 - 5x_2 \end{aligned}$$

The corresponding SFG is shown in Fig. 7.2, where single circles refer to nonbasic variables and constant input nodes, and double circles refer to basic variables which must be solved for. Figure 7.2 shows that the solution at this step is

$$\begin{aligned} Z &= 0, & x_3 &= 10 \\ x_1 &= 0, & x_4 &= 40 \\ x_2 &= 0, & x_5 &= 40 \end{aligned}$$

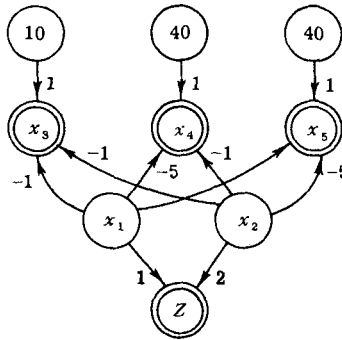
The basic variable in relation (3) is designated as  $x_2$  in place of leaving basic variable  $x_5$ .

*Step 2.* The basic variables are now  $x_3(1)$ ,  $x_4(2)$ , and  $x_2(3)$ . Figure 7.3 shows the following results:

$$\begin{aligned} Z &= 16, & x_3 &= 2 \\ x_1 &= 0, & x_4 &= 32 \\ x_2 &= 8, & x_5 &= 0 \end{aligned}$$

$x_1$  is now assigned as the basic variable in relation (1); the leaving variable is  $x_3$ .

$$\begin{aligned} Z &= x_1 + 2x_2 \\ x_3 &= -x_1 - x_2 + 10 \\ x_4 &= -5x_1 - x_2 + 40 \\ x_5 &= -x_1 - 5x_2 + 40 \\ \text{Set } x_1 &= x_2 = 0. \end{aligned}$$



Solution for step 1:

$$\begin{aligned} x_3 &= (T_{10 \rightarrow x_3} \times 10) + (T_{40 \rightarrow x_3} \times 40) + (T_{40 \rightarrow x_3} \times 40) \\ &\quad + (T_{x_1 \rightarrow x_3} \times x_1) + (T_{x_2 \rightarrow x_3} \times x_2) \\ &= (1 \times 40) + (0 \times 40) + (0 \times 40) + (-1 \times 0) + (-1 \times 0) = 10 \\ x_4 &= 1 \times 40 = 40, \\ x_5 &= 1 \times 40 = 40 \\ Z &= (1 \times 0) + (2 \times 0) = 0 \end{aligned}$$

Entering and leaving variables:

$$\begin{aligned} C'_M &= \begin{cases} T_{x_1 \rightarrow Z} = 1 \\ T_{x_2 \rightarrow Z} = 2 \end{cases} \leftarrow x_2 \text{ enters} \\ -a'_{13} &= T_{x_2 \rightarrow x_3} = -1, & b'_1/a'_{13} &= 10/[-(-1)] = 10 \\ -a'_{24} &= T_{x_2 \rightarrow x_4} = -1, & b'_2/a'_{24} &= 40/[-(-1)] = 40 \\ -a'_{35} &= T_{x_2 \rightarrow x_5} = -5, & b'_3/a'_{35} &= 40/[-(-5)] = 8 \leftarrow x_5 \text{ leaves} \end{aligned}$$

Fig. 7.2. LP example; step 1.

Step 3. The basic variables are now  $x_1(1)$ ,  $x_4(2)$  and  $x_2(3)$ . Figure 7.4 shows the following results

$$\begin{aligned} Z &= 17.5, & x_3 &= 0 \\ x_1 &= 2.5, & x_4 &= 20 \\ x_2 &= 7.5, & x_5 &= 0 \end{aligned}$$

and

$$C'_3 = -\frac{3}{4}, \quad C'_5 = -\frac{1}{4}$$

Both  $C'_3$  and  $C'_5$  are negative; therefore, this is the optimal solution.

In this example the SFG's are drawn at each iteration to show the procedure; it is not necessary in practice. The Mason computer routine needs

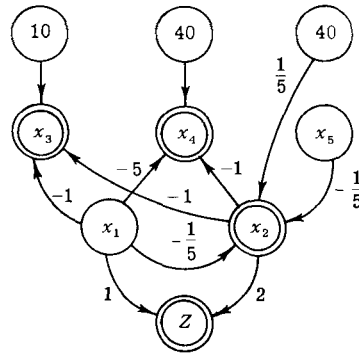
$$Z = x_1 + 2x_2$$

$$x_3 = -x_1 - x_2 + 10$$

$$x_4 = 5x_1 - x_2 + 40$$

$$x_2 = -\frac{1}{5}x_1 - \frac{1}{5}x_5 + \frac{1}{5}(40)$$

$$\text{Set } x_1 = x_5 = 0.$$



*Solution for step 2:*

$$x_3 = (1 \times 10) + (-1)(\frac{1}{5})(40) = 10 - 8 = 2 = b_1'$$

$$x_4 = (1 \times 40) + (-1)(\frac{1}{5})(40) = 40 - 8 = 32 = b_2'$$

$$x_2 = (\frac{1}{5} \times 40) = 8 = b_3'$$

$$Z = (2)(\frac{1}{5})(40) = 16$$

*Entering and leaving variables:*

$$C'_M = \begin{cases} T_{x_1 \rightarrow Z} = 1 & \leftarrow x_1 \text{ enters} \\ T_{x_5 \rightarrow Z} = (-\frac{1}{5})(2) = -\frac{2}{5} \end{cases}$$

$$-a'_{11} = T_{x_1 \rightarrow x_3} = (-1) + (\frac{1}{5})(-1) = -\frac{6}{5}, \quad b_1'/a'_{11} = 2/\frac{6}{5} = 10/4 \leftarrow x_3 \text{ leaves}$$

$$-a'_{21} = T_{x_1 \rightarrow x_4} = (-5) + (-\frac{1}{5})(-1) = 4(\frac{4}{5}), \quad b_2'/a'_{21} = 32/\frac{16}{5} = 160/24$$

$$-a'_{31} = T_{x_1 \rightarrow x_2} = -\frac{1}{5}, \quad b_3'/a'_{31} = 8/\frac{1}{5} = 40$$

**Fig. 7.3.** LP example; step 2.

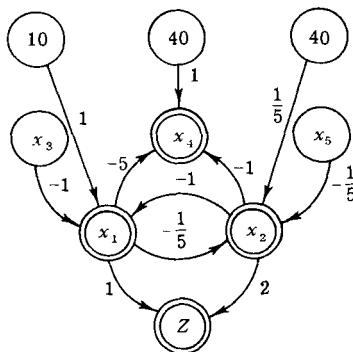
as input data all the branches in such a way that starting node number, ending node number, and gain are specified for each branch.

Therefore, the necessary data for each iteration are a row vector  $(C_1, \dots, C_n)$ , a column vector  $(b_1, \dots, b_n)$  and an  $(n)(m+n)$  matrix  $[A, I]$ , where  $[A, I]$  has the form

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} & 1 & 0 \\ \vdots & & \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} & 0 & 1 \end{bmatrix}$$

Suppose  $JB(I)$  is the designation of a basic variable specified in relation I, i.e. variable  $JB(I)$  is not eliminated in relation I through the Gauss-Jordan elimination; then, all the branches can be specified. In relation I, the starting

$$\begin{aligned} Z &= x_1 + 2x_2 \\ x_1 &= -x_2 - x_3 + 10 \\ x_4 &= -5x_1 - x_2 + 40 \\ x_2 &= -\frac{1}{5}x_1 - \frac{1}{5}x_5 + \frac{1}{5}(40) \\ \text{Set } x_3 &= x_5 = 0. \end{aligned}$$



Solution for step 3:

$$x_1 = \left[ \frac{1}{1 - (-1)(-\frac{1}{5})} \right] (10) + \left[ \frac{(\frac{1}{5})(-1)}{1 - (-1)(-\frac{1}{5})} \right] (40) = 2.5$$

$$x_2 = \left[ \frac{\frac{1}{5}}{1 - (-1)(-\frac{1}{5})} \right] (40) + \left[ \frac{(1)(-\frac{1}{5})}{1 - (-1)(-\frac{1}{5})} \right] (10) = 7.5$$

$$\begin{aligned} x_4 &= \left[ \frac{(1)(1 - (-1)(-\frac{1}{5}))}{1 - (-1)(-\frac{1}{5})} \right] (40) + \left[ \frac{(1)(-5)}{1 - (-1)(-\frac{1}{5})} \right] (10) + \left[ \frac{(1)(-\frac{1}{5})(-1)}{1 - (-1)(-\frac{1}{5})} \right] (10) \\ &\quad + \left[ \frac{(-1)(\frac{1}{5})}{1 - (-1)(-\frac{1}{5})} \right] (40) + \left[ \frac{(\frac{1}{5})(-1)(-5)}{1 - (-1)(-\frac{1}{5})} \right] (40) = 20 \end{aligned}$$

$$\begin{aligned} Z &= \left[ \frac{1}{1 - (-1)(-\frac{1}{5})} \right] \{ [(1)(1) + (1)(-\frac{1}{5})(2)] (10) \\ &\quad + [(\frac{1}{5})(2) + (\frac{1}{5})(-1)(1)] (40) \} = 17.5 \end{aligned}$$

Entering and leaving variables:

$$C'_M = \begin{cases} T_{x_3 \rightarrow Z} = \frac{1}{1 - (-1)(-\frac{1}{5})} [(-1)(1) + (-1)(-\frac{1}{5})(2)] = -\frac{3}{5} \\ T_{x_5 \rightarrow Z} = \frac{1}{1 - (-1)(-\frac{1}{5})} [(-\frac{1}{5})(2) + (-\frac{1}{5})(-1)(1)] = -\frac{1}{5} \end{cases}$$

Fig. 7.4. LP example: step 3 (final).

nodes are  $b_I$  and  $x_J$  where  $J$  is 1 to  $(n + m)$  and  $a_{IJ} \neq 0$ . The ending node is  $x_{JB(I)}$  and the branch gain is  $(-a_{IJ}/a_{I,JB(I)})$ .

The procedures described have been formulated as a computer program and a number of example problems were solved [1]. The computer programs, which are described in Appendix J, are available from the authors.

**Discussion of Solution Procedure (MASNLP)**

The SFG algorithm called MASNLP given in Appendix J solves any LP problem, provided that input data are given in standard LP problem form.

One of the difficulties with the method is the necessity of redrawing the SFG at each iteration ("redrawing" does not mean drawing a visual SFG). This was resolved simply by keeping the A matrix and the number of the basic variable  $JB(I)$ .

Another difficulty is that the coefficient of the basic variable  $x_{JB(I)}$  in relation I may be zero. A simple example is given in Table 7.2. This can be handled by adding both sides of an equation, which has nonzero coefficient for  $x_{JB(I)}$ , to relation I.

For example, problems 1 and 2 are exactly identical.

**Problem 1**

$$\begin{array}{rcl} Z - 2x_1 - x_2 & & = 0 \\ x_1 - x_2 + x_3 & & = 2 \\ x_1 & + x_4 & = 5 \\ & x_2 & + x_5 = 5 \end{array}$$

**Problem 2**

$$\begin{array}{rcl} Z - 2x_1 - x_2 & & = 0 \\ x_1 - x_2 + x_3 & & = 2 \\ 2x_1 - x_2 + x_3 + x_4 & & = 7 \\ x_1 & + x_3 & + x_5 = 7 \end{array}$$

**An Advantage of MASNLP**

The simplex algorithm is commonly implemented by a matrix calculation procedure called the revised simplex method. This method has an advantage insofar as it avoids matrix inversion procedures. On the other hand, it wastes large amounts of computer memory by storing zero elements, and, in the case of sparse matrices, it does a large number of zero multiplications.

The SFG, on the other hand, picks up only nonzero elements. This implies that the MASNLP algorithm can be superior to the revised simplex when there are a large number of zero elements.

TABLE 7.2  
CASE OF ZERO COEFFICIENT ASSIGNED AS A BASIC VARIABLE

Problem: Maximize

$$Z = 2x_1 + x_2$$

Subject to

$$\begin{aligned} x_1 - x_2 &\leq 2, & x_1 &\leq 5, \\ x_2 &\leq 5, & x_1, x_2 &\leq 0. \end{aligned}$$

Standard form:

$$\begin{aligned} Z - 2x_1 - x_2 &= 0, \\ x_1 - x_2 + x_3 &= 2, \\ x_1 &+ x_4 = 5, \\ x_2 &+ x_5 = 5. \end{aligned}$$

Simplex tableau:

Iteration	Relation	Z	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	b
1.	(0)	1	-2	-1	0	0	0	0
	(1)	0	1	-1	1	0	0	2
	(2)	0	1	0	0	1	0	5
	(3)	0	0	1	0	0	1	5
2.	(0)	1	0	-3	2	0	0	4
	(1)	0	1	-1	1	0	0	2
	(2)	0	0	1	-1	1	0	3
	(3)	0	0	1	0	0	1	5
3.	(0)	1	0	0	-1	3	0	13
	(1)	0	1	0	0	0	0	5
	(2)	0	0	1	-1	1	0	3
	(3)	0	0	0	1	-1	1	2
4.	(0)	1	0	0	0	2	1	15
	(1)	0	1	0	0	0	0	5
Optimum	(2)	0	0	1	0	0	1	5
	(3)	0	0	0	1	-1	1	2

Comment:

1. At step 2,  $x_2$  is assigned as basic variable in relation (2), but  $a_{22} = 0$  in standard form.

2. At step 3,  $x_3$  is assigned as a basic variable in relation (3) but  $a_{33} = 0$  in standard form.





This is the way to make the decrease in  $b_0'$  a minimum when  $x_l$  in relation (0) of Eq. (7.7) is eliminated. Then (a) and (b) are continued until all the  $b_j'$ 's, ( $j = 1, \dots, m$ ) become positive.

In the analogous SFG method, once the nonbasic and basic variables are assigned,  $x_1, \dots, x_n$  and  $x_{n+1}, \dots, x_{m+n}$ , respectively, the SFG is drawn with  $x_1, \dots, x_n$  as input nodes and with  $x_{n+1}, \dots, x_{m+n}$  and  $Z$  as output nodes using the original equations as in the ordinary MASNLP algorithm. Then  $C_j'$  and  $b_i'$  are calculated,  $k$  is chosen,  $a'_{ki}$  is calculated as  $(-T_{x_i \rightarrow x_{n+k}})$ , and  $l$  is chosen as in the dual simplex algorithm.

What is the significance of using the SFG to solve the dual simplex problem? Suppose some of the parameters of the original problem, the  $a_{ij}$ 's,  $b_i$ 's, and  $C_j$ 's have been changed. Then, we must change the branch gains of the final SFG in accordance with the change in the parameters and repeat the computation.

If all the  $C_j'$  ( $j = 1, \dots, n$ ) and the  $b_i'$  ( $i = 1, \dots, m$ ), which were calculated by use of the transmittances, or as the transmittances, are positive, the solution is still optimal, but if all the  $C_j'$ 's are not, then we continue the MASNLP procedure; if all the  $b_j'$ 's are not positive, then we go to the dual simplex by the SFG method.

## Postoptimal Analysis

The final simplex tableau can be used to analyze how a change in the original parameters affects the optimal solution.

Five cases normally occur: (a) change in  $C_j$ ; (b) change in  $b_i$ ; (c) change in  $a_{ij}$ ; (d) addition of a new constraint; and (e) addition of a new variable.

It is possible to make all five of these postoptimal analysis without changing the basic procedure of the SFG method. In cases (a), (b), and (c), the appropriate branch gain may be changed and each necessary  $C_j'$ ,  $b_i'$ , and  $a'_{ij}$  can be recalculated using the final SFG. Then optimality will be judged as usual, and if the solution is not optimal or not feasible, ordinary MASNLP or dual simplex procedures by SFG can be followed. In cases (d) and (e), new nodes and branches are added to the final SFG, noting that in case (d), the new slack variable is a basic variable, and in case (e), the new variable is nonbasic. The ordinary calculation procedure can then be followed. Thus the convenient applicability of the SFG method to post optimal analysis is an advantage of this method.

### Sensitivity Analysis

Parametric linear programming is a method whereby some of the parameters and the sensitivity of each parameter are changed stepwise. The optimum values of the parameters are also studied. This is nothing more than a sensitivity analysis of the linear system itself, and it will now be briefly demonstrated how the sensitivity procedures developed in Chapter 5 can be applied.

The basic idea of doing LP by SFG is restated as

$$\begin{aligned} x_j^B &= \sum_{i=1}^m [(T_{b_i \rightarrow x_j^B})b_i] = b_i' \\ Z &= \sum_{i=1}^m [(T_{b_i \rightarrow Z})b_i] \\ C_j' &= T_{x_j^N \rightarrow Z} \\ a_{ij}' &= -T_{x_j^N \rightarrow x_j^B} \end{aligned} \quad (7.8)$$

where  $x_j^B$  is a basic variable which is in relation (i), and  $x_j^N$  is a non-basic variable. Then

$$\begin{aligned} \partial x_j^B / \partial b_i &= \partial b_i' / \partial b_i = T_{b_i \rightarrow x_j^B} \\ \partial x_j^B / \partial a_{ij} &= \partial b_i' / \partial a_{ij} = \sum_{i=1}^n [(\partial T_{b_i \rightarrow x_j^B} / \partial a_{ij})b_i] \end{aligned} \quad (7.9)$$

and a similar equation for  $Z$ ,  $C_j'$  and  $a_{ij}'$  exists.

The sensitivity  $\partial T_{i \rightarrow j} / \partial p$ , where  $p$  is a gain of a branch connecting node  $m$  and  $n$ , and  $i$  and  $j$  are input and output, can be computed as

$$\partial T_{j \rightarrow i} / \partial p = T_{j \rightarrow m} \cdot T_{f \rightarrow i} \quad (7.10)$$

where  $f$  is a fictitious input node connected with node  $n$  having a gain of 1. Therefore all the desired sensitivities can be calculated by applying Eqs. (7.9) and (7.10) directly, using the final SFG.

### Nonlinear Systems

Direct application of SFGs is limited to linear systems. But, if nonlinear systems are linearized, nonlinear optimization problems with nonlinear constraints may also be solved effectively.

Consider the following nonlinear optimization problem with nonlinear constraints:

Maximize

$$J = f(x_1, \dots, x_n) \quad (7.11)$$

Subject to

$$g_i(x_1, \dots, x_n) = 0, \quad i = 1, \dots, m, \quad m < n$$

Total derivatives of these equations are

$$dJ = (\partial f / \partial x_1) dx_1 + (\partial f / \partial x_2) dx_2 + \dots + (\partial f / \partial x_n) dx_n \quad (7.12)$$

$$dg_i = (\partial g_i / \partial x_1) dx_1 + (\partial g_i / \partial x_2) dx_2 + \dots + (\partial g_i / \partial x_n) dx_n = 0 \quad (7.13)$$

The number of equations  $(m + 1)$  is less than the number of variables,  $(n + 1)$  including  $J$ , in properly defined optimization problems. Then one can choose  $(n - m)$  independent variables.

Suppose  $(n - m)$  variables  $x_1, \dots, x_{n-m}$  are chosen as the independent variables. Then,  $m$  of the  $g_i$  equations  $i = 1, \dots, m$ , can be solved for  $x_{n-m+1}, \dots, x_n$  and substituted into  $f(x_1, \dots, x_n)$ . We obtain

$$Z = F(x_1, \dots, x_{n-m}) \quad (7.14)$$

Equation (7.14) is an alternate form of the original problem where the constraints are incorporated into the objective function.  $F(x)$  at the optimum is stationary in terms of the variables not eliminated by the substitution

$$\partial F / \partial x_1 = \partial F / \partial x_2 = \dots = \partial F / \partial x_{n-m} = 0 \quad (7.15)$$

and the second derivatives

$$\partial^2 F / \partial x_1^2, \partial^2 F / \partial x_2^2, \dots, \partial^2 F / \partial x_{n-m}^2$$

are all negative.

Unfortunately the  $g_i$ 's are not always simple and the number of variables may be large so this direct substitution approach to optimization is frequently not feasible in practical situations. However, it is always possible to write  $dJ$  and  $dg_i$ , the total derivatives, as in Eqs. (7.12) and (7.13) and  $\partial f / \partial x_j$  and  $\partial g_i / \partial x_j$ , ( $i = 1, \dots, m$ ;  $j = 1, \dots, n$ ) may generally be obtained analytically. The individual partial derivatives at point  $k$ ,  $(\partial f / \partial x_1, \dots, \partial f / \partial x_n)_k$ , define a direction in space called the *gradient* at that point. The objective function  $f$  increases most rapidly in its gradient direction, this

direction being perpendicular to the *contour tangent* at point  $k$ . A contour is defined such that  $f$  is constant along it.

Equations (7.12) and (7.13) must be rewritten in order to draw an appropriate SFG.

$$dJ = \left( \frac{\partial f}{\partial x_1} \right) dx_1 + \cdots + \left( \frac{\partial f}{\partial x_n} \right) dx_n \quad (7.16)$$

$$\begin{aligned} dx_{n-m+1} &= \frac{-1}{(\partial f / \partial x_{n-m+1})} \left[ \left( \frac{\partial f}{\partial x_1} \right) dx_1 + \cdots + \left( \frac{\partial f}{\partial x_{n-m}} \right) dx_{n-m} \right. \\ &\quad \left. + \left( \frac{\partial f}{\partial x_{n-m+2}} \right) dx_{n-m+2} + \cdots + \left( \frac{\partial f}{\partial x_n} \right) dx_n \right] \\ &\vdots \\ dx_n &= - \frac{1}{(\partial f / \partial x_n)} \left[ \left( \frac{\partial f}{\partial x_1} \right) dx_1 + \cdots + \left( \frac{\partial f}{\partial x_{n-1}} \right) dx_{n-1} \right] \end{aligned} \quad (7.17)$$

The SFG looks like Fig. 7.5. It can be seen that  $dJ$  is the sum of products;  $T_{dx_i \rightarrow dJ} dx_i$  ( $i = 1, \dots, n-m$ ), that is,

$$dJ = T_{dx_1 \rightarrow dJ} dx_1 + \cdots + T_{dx_{n-m} \rightarrow dJ} dx_{n-m} \quad (7.18)$$

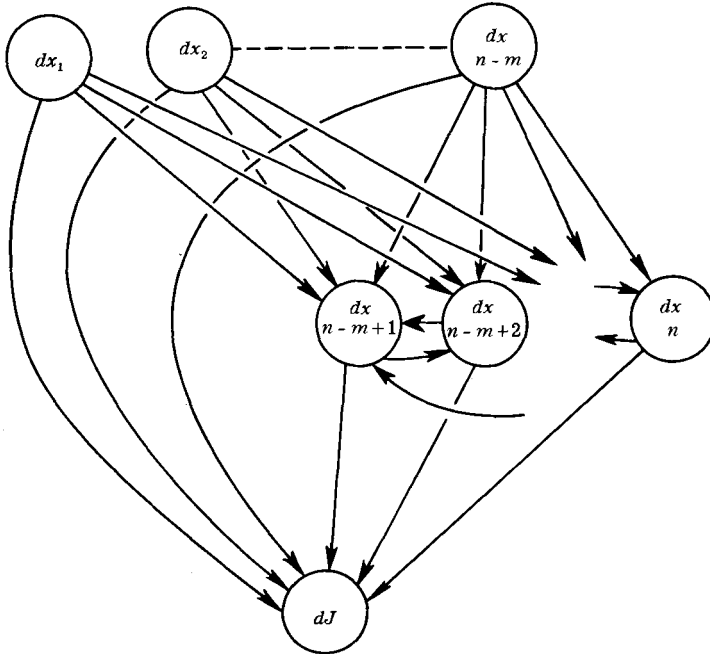


Fig. 7.5. SFG for Eq. (7.17).

Then

$$\partial F / \partial x_i = T_{dx_i \rightarrow dJ}, \quad i = 1, \dots, n - m \quad (7.19)$$

follows where

$$J = F(x_1, x_2, \dots, x_{n-m})$$

## Method of Feasible Directions

The method of feasible directions (MFD) for solving nonlinear programming problems with inequality constraints was developed by Zoutendijk [3]. The algorithm begins with any point in the feasible region. A feasible direction of movement is then chosen. Since it is frequently impossible to move along the gradient, Zoutendijk chooses to move in that feasible direction which makes the smallest possible angle with the gradient. This direction is found in the solution to a linear programming problem in which the increase in the objective function is maximized, subject to feasibility constraints. This direction is then followed, a search method being used to find a new base point which either lies on a constraint, or represents the furthestest distance one can travel in the feasible direction and still cause the objective function to increase.

The MFD works very well if the constraints are linear, i.e., of the form

$$g_i(\mathbf{x}) = \sum_{j=1}^n a_{ij}x_j - b_i$$

To demonstrate, let us assume that at step  $k$  of the procedure we are in the feasible region. The gradient vector  $(\partial f_1 / \partial x_1, \dots, \partial f / \partial x_n)_k$  is then calculated and we proceed by a conventional gradient search.

1. Find  $\lambda$  so that  $f(\mathbf{x}) = f(\lambda)$  is a maximum (or minimum) in the feasible region. The  $(k + 1)$  vector of unknowns in terms of  $\lambda$  is

$$\mathbf{x}^{k+1} = \mathbf{x}^k + (\partial f / \partial x_1, \dots, \partial f / \partial x_n)_k^T \lambda \quad (7.20)$$

For a minimization problem the plus sign is changed to a minus. The second term on the right-hand side is equivalent to  $(dx_1, \dots, dx_n)^T$ , the change in the variables along the gradient direction.

2. The magnitude of  $\lambda$  is established by a one-dimensional search technique along the gradient using the original objective function which, at point  $k$ , is a function only of  $\lambda$ ,  $f(\mathbf{x}) = f(\lambda)$ . The Fibonacci search technique described in Appendix G may be used.

$\mathbf{x}^{k+1}$  may be in the feasible region, in which case we go to (1). If it is on the boundary of the feasible region, we go to 3.

3.  $\mathbf{x}^k$  is on the boundary. Hence

$$\begin{aligned} \sum_{j=1}^n a_{ij}x_j^k &= b_i, & i = 1, \dots, l \leq m \\ \sum_{j=1}^n a_{ij}x_j^k &< b_i, & i = (l+1), \dots, m \\ x_j^k &= 0, & j = 1, \dots, l' \leq n \\ x_j^k &> 0, & j = (l'+1), \dots, n \end{aligned} \quad (7.21)$$

At the next iteration

$$\sum_{j=1}^n a_{ij}x_j, \quad i = 1, \dots, l \leq m$$

should not be increased, while

$$\sum_{j=1}^n a_{ij}x_j, \quad i = (l+1), \dots, m$$

may be increased, and  $x_j, j = 1, \dots, l'$  should not be decreased. We can no longer follow the gradient, so we proceed along a direction vector  $\mathbf{r} = (dx_1, \dots, dx_n)^T \lambda$  constrained as follows:

$$\begin{aligned} \sum_{j=1}^n a_{ij}x_j &= \sum_{j=1}^n a_{ij}(x_j^k + r_j) \leq b_i, & i = 1, \dots, l \\ x_j^{k+1} &= x_j^k + r_j \geq 0, & i = 1, \dots, l' \end{aligned}$$

but

$$\sum_{j=1}^n a_{ij}x_j^k = b_i, \quad i = 1, \dots, l$$

and

$$x_j^k = 0, \quad j = 1, \dots, l'$$

Thus the constraints become

$$\begin{aligned} \sum_{j=1}^n a_{ij}r_j &\leq 0, & i = 1, \dots, l \\ r_j &\geq 0, & j = 1, \dots, l' \end{aligned} \quad (7.22)$$

We are looking for the direction in which  $f(\mathbf{x})$  will be increased most; in other words, we are looking for a maximum

$$df = (\partial f / \partial x_1) dx_1 + \cdots + (\partial f / \partial x_n) dx_n$$

Thus, the LP problem is: find the direction vector  $\mathbf{r}$ , which will

Maximize

$$Z = (\partial f / \partial x_1) r_1 + (\partial f / \partial x_2) r_2 + \cdots + (\partial f / \partial x_n) r_n \quad (7.23)$$

Subject to

$$\begin{aligned} \sum_{j=1}^n a_{ij} r_j &\leq 0, & i = 1, \dots, l \\ r_j &\geq 0, & j = 1, \dots, l' \\ 1 &\geq r_j \geq -1, & j = 1, \dots, n. \end{aligned}$$

The  $\pm 1$  constraints are used instead of the normalization,  $\sum r_j^2 = 1$  to permit a LP formulation. This may increase or decrease the number of iterations to reach the optimum.

Three two-dimensional cases are given in Fig. 7.6. In case (I), ① goes to ② by the ordinary gradient method. The gradient vector ③ at ② is outside of the feasible region; then the LP solution gives ④, and ⑤ is obtained where the solution to the LP problem is zero. In case (II), the gradient vector at ①, which is ② is outside the feasible region. The LP solution gives ③ where the straight line ③ → ⑥ touches the feasible LP region. Then the next iteration point ⑦ is obtained by search methods. If a constraint such as “the norm of the solution vector = 1,” has been used, the LP problem would be a quadratic programming problem. This would have given a solution vector on the boundary and the final solution could have been obtained with fewer iterations. In case (III), conversely, the  $-1 \leq r \leq 1$  constraint reduces the number of iterations.

4. After the LP problem is solved for  $\mathbf{r}$ , we again obtain the maximum feasible  $f(x) = f(\lambda)$  by a Fibonacci search along the constraint boundary.

5. If the direction vector  $\mathbf{r}$ , or  $J$  are zero in step 3, optimality has been achieved. If not, we return to step 1 if we are in the feasible region, or to step 3 if we are on a boundary.

The optimality condition can be interpreted geometrically. Suppose the  $n$ -dimensional gradient vector is outwardly directed and orthogonal to hyperplane  $R$ . Then the inner product of the gradient vector and any vector which is directed to another side of  $R$  is negative. Thus, if the  $\mathbf{r}$ -vector



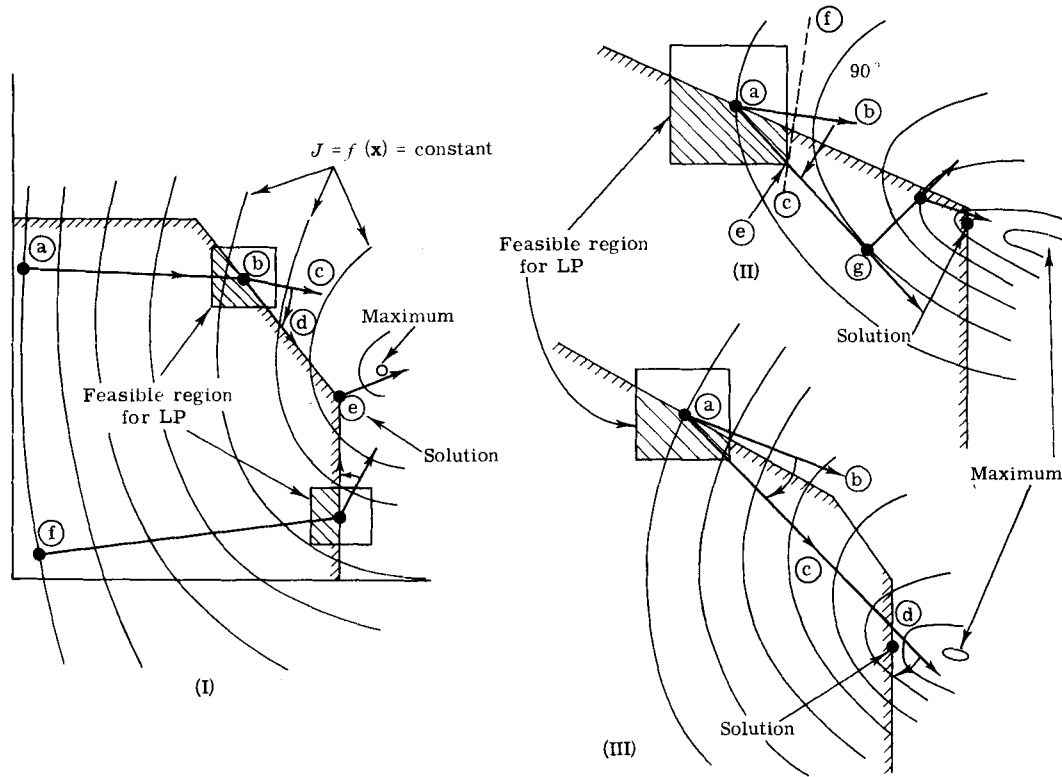


Fig. 7.6. Some examples of the method of feasible direction in two dimensions.

cannot be directed to the same side of  $R$  as the gradient vector, the  $Z$  is at most equal to zero.

The MFD procedure will now be demonstrated with a numerical example:

Maximize

$$f = x_1 x_2, \quad (7.24)$$

Subject to

$$x_1 + 2x_2 \leq 10, \quad x_1 \leq 7, \quad x_1, x_2 \geq 0 \quad (7.25)$$

### First Iteration

As a first step we establish the gradient direction and move as far as possible towards an optimum without violating a constraint. The search is started at (1, 1), which is point (1) in Fig. 7.7.

$$\partial f / \partial x_1 = 1, \quad \partial f / \partial x_2 = 1$$

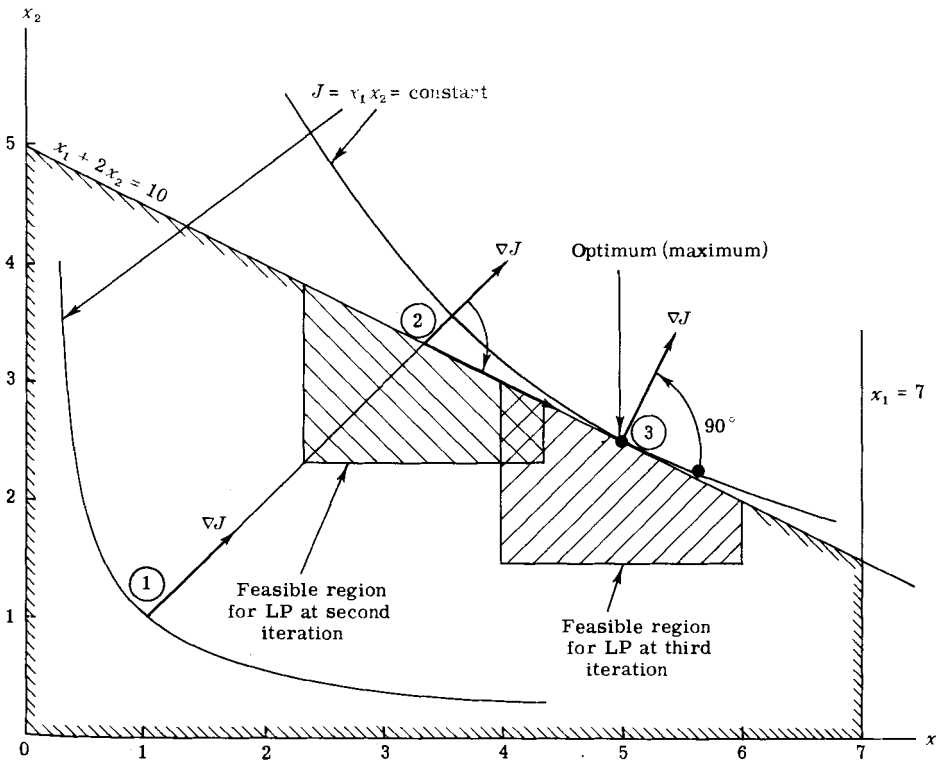


Fig. 7.7. Example for method of feasible direction.

By Eq. (7.20)

$$x_1^2 = x_1^1 + (1)\lambda, \quad x_2^2 = x_2^1 + (1)\lambda$$

The constraint equations are used to establish the maximum distance of movement, because  $f$  increases continuously as  $\lambda$  increases. Thus a Fibonacci search is unnecessary. Since

$$f^2 = (x_1' + \lambda)(x_2' + \lambda)$$

then

$$\partial f / \partial \lambda = 2(\lambda + 1)$$

By Eq. (7.25)

$$(1 + \lambda) + 2(1 + \lambda) \leq 10, \quad (1 + \lambda) \leq 7$$

The maximum  $\lambda$  is  $\frac{7}{3}$ , thus

$$x_1^2 = 1 + \frac{7}{3} = \frac{10}{3}, \quad x_2^2 = 1 + \frac{7}{3} = \frac{10}{3}$$

This is point (2) on Fig. 7.7. We are now on a constraint boundary, so we go to step 3.

### Second Iteration

We establish the feasible direction vector  $\mathbf{r}$  by solving the LP problem:

Maximize

$$Z = (\partial f / \partial x_1)r_1 + (\partial f / \partial x_2)r_2 = \frac{10}{3}r_1 + \frac{10}{3}r_2$$

Subject to Eq. (7.23)

$$r_1 + 2r_2 \leq 0, \quad -1 \leq r_1, \quad r_2 \leq 1$$

The solution is  $r_1 = 1$ ,  $r_2 = -\frac{1}{2}$ ,  $Z = \frac{5}{3}$ . Since  $Z$  is positive, this is not an optimum. It is now necessary to find the distance along  $\mathbf{r}$  to move.

First we establish the maximum distance  $\lambda$ ; we can then move without violating a constraint.

$$x_1^3 = x_1^2 + \lambda = \frac{10}{3} + \lambda$$

$$x_2^3 = x_2^2 + \lambda = \frac{10}{3} - \frac{1}{2}\lambda$$

By Eq. (7.25)

$$(\frac{10}{3} + \lambda) + 2(\frac{10}{3} - \frac{1}{2}\lambda) = 10, \quad (\frac{10}{3} + \lambda) \leq 7$$

Thus the maximum  $\lambda$  is  $\frac{11}{3}$ . It is possible, however, that the optimum has already been passed. A one-dimensional Fibonacci search may be used. This is given as an example in Appendix G. Alternatively, a test of the objective function can be made:

$$f = (\frac{10}{3} + \lambda)(\frac{10}{3} - \lambda),$$

$$\partial f / \partial \lambda = 0 = \frac{5}{3} - \lambda, \quad \lambda = \frac{5}{3}$$

Therefore, the smaller  $\lambda$  is used and

$$x_1^3 = \frac{10}{3} + \frac{5}{3} = 5, \quad x_2^3 = \frac{10}{3} - (\frac{1}{2})(\frac{5}{3}) = 2.5, \quad f = 12.5$$

This is point (3) on Fig. 7.7.

### Third Iteration

We are still on the boundary so it is necessary to obtain the solution to the LP problem.

Maximize

$$Z = 2.5r_1 + 5r_2$$

Subject to

$$r_1 + 2r_2 \leq 0, \quad -1.0 \leq r_1, \quad r_2 \leq 1.0$$

The solution is  $r_1 = 0$ ,  $r_2 = 0$ ,  $Z = 0$ . Since  $Z$  is not positive,  $\mathbf{x}^3$  is the optimum.

It has been shown that the MFD works very well if the constraints are linear. If the constraints are nonlinear, the LP part of the solution procedure does not work. Zoutendijk recommends that, if the constraints are nonlinear, linearize the constraints necessary for the LP part and move the iteration point so that the next point does not go out of the feasible region too much, and then repeat the procedure.

## A Heat Exchanger Problem

The heat exchanger system with nine heat exchangers discussed in Chapter 6 was optimized by Takamatsu and co-workers [4] by a multilevel technique and conjugate gradient methods.

*Problem.* Obtain the cold and hot stream temperatures,  $t$ 's and  $T$ 's, so that the total heat exchanger area  $A$  is a minimum. The configuration of heat exchangers and given conditions are shown in Fig. 7.8.

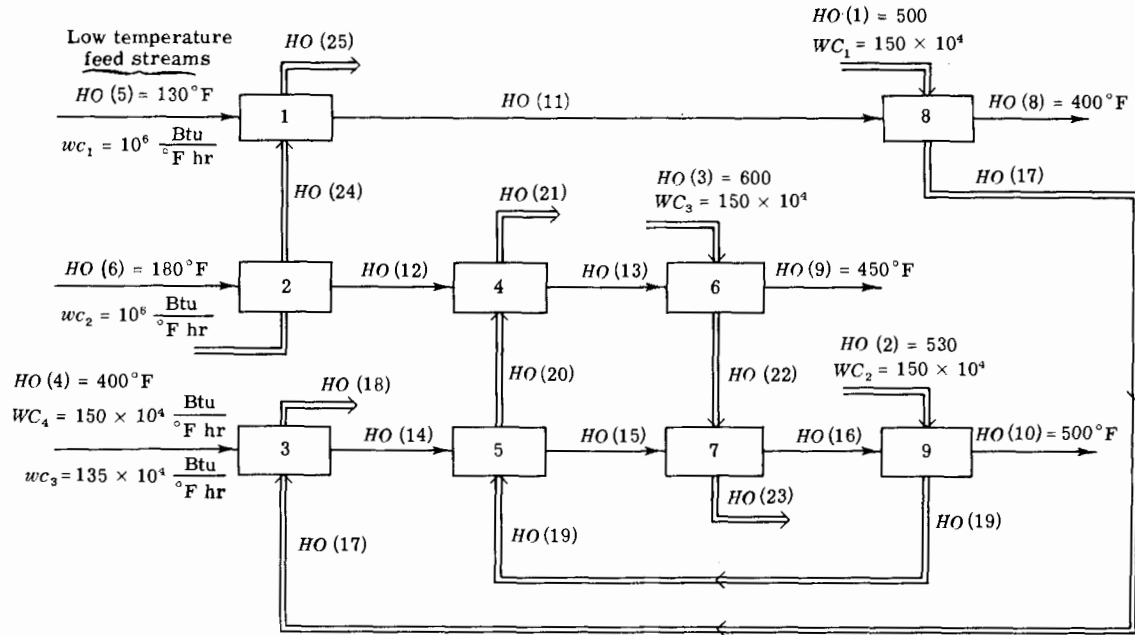


Fig. 7.8. System of nine heat exchangers. HO(1), ..., HO(10): given temperature; HO(11), ..., HO(25): variable temperature; HO(26), ..., HO(34): heat exchanger area.

This problem is formulated as follows:

Objective function:

$$J = \sum_{k=1}^9 A_k \quad (\text{minimize})$$

Heat balance:

$$wc_k(t_0^k - t_i^k) = WC_k(T_i^k - T_0^k)$$

Heat transfer:

$$WC_k(T_i^k - T_0^k) = U_k A_k \frac{(T_0^k - t_i^k) - (T_i^k - t_0^k)}{\ln[(T_0^k - t_i^k)/(T_i^k - t_0^k)]}$$

Constraints for  $t$ 's and  $T$ 's:

$$t_0^k \geq t_i^k, \quad T_i^k \geq T_0^k, \quad T_i^k > t_0^k, \quad T_0^k > t_i^k$$

where  $k = 1, \dots, 9$ .

There are 9 linear and 9 nonlinear equality constraints, and 36 linear inequality constraints.

### **Solution to the Heat Exchanger System**

The solution procedure was developed using a four-exchanger system which is simpler than Takamatsu's but has the same characteristics. Figure 7.9 shows the simpler problem, which is formulated as follows:

Objective function:

$$J = A_1 + A_2 + A_3 + A_4 \quad (0)$$

Heat balance:

$$wc_1(t_1 - t_0) = WC_1(T_0 - T_1) \quad (1)$$

$$wc_1(t_2 - t_1) = WC_2(T_3 - T_4) \quad (2)$$

$$wc_2(t_4 - t_3) = WC_1(T_1 - T_2) \quad (3)$$

$$wc_2(t_5 - t_4) = WC_2(T_4 - T_5) \quad (4)$$

(7.26)

Heat transfer:

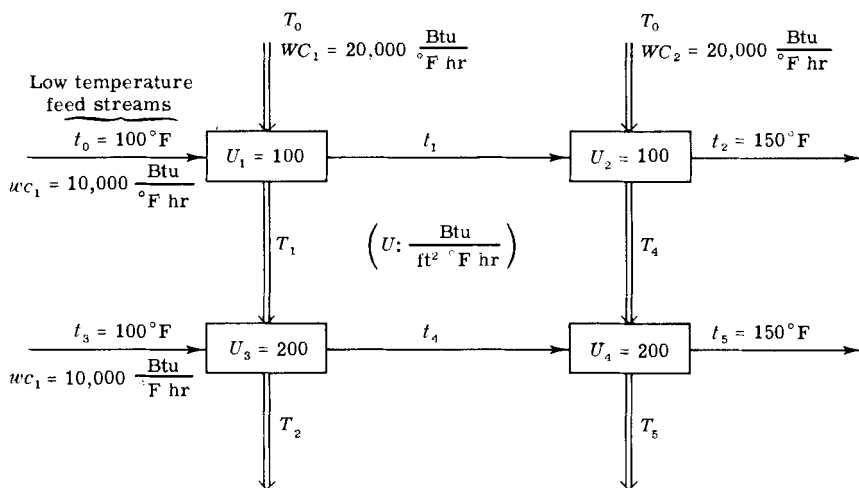
$$WC_1(T_0 - T_1) = U_1 A_1 \frac{(T_0 - t_1) - (T_1 - t_0)}{\ln[(T_0 - t_1)/(T_1 - t_0)]} \quad (5)$$

$$WC_2(T_3 - T_4) = U_2 A_2 \frac{(T_3 - t_2) - (T_4 - t_1)}{\ln[(T_3 - t_2)/(T_4 - t_1)]} \quad (6)$$

$$WC_1(T_1 - T_2) = U_3 A_3 \frac{(T_1 - t_4) - (T_2 - t_3)}{\ln[(T_1 - t_4)/(T_2 - t_3)]} \quad (7)$$

$$WC_2(T_4 - T_5) = U_4 A_4 \frac{(T_1 - t_4) - (T_2 - t_3)}{\ln[(T_1 - t_4)/(T_2 - t_3)]} \quad (8)$$

The nomenclature is:  $A$  = heat exchanger area;  $t$  = cold stream temperature;  $U$  = heat transfer area (constant);  $wc$  = heat flow rate, cold stream (constant); and  $WC$  = heat flow rate, hot stream (constant).



	$T_0$ ( $^\circ\text{F}$ )	$T_3$ ( $^\circ\text{F}$ )
Case I	150	200
II	150	180
III	180	180

Fig. 7.9. A system of four heat exchangers.

Temperature constraints are

$$t_0 \leq t_1 \leq t_2 \quad (9)$$

$$t_3 \leq t_4 \leq t_5 \quad (10)$$

$$T_2 \leq T_1 \leq T_0 \quad (9')$$

$$T_5 \leq T_4 \leq T_3 \quad (10')$$

$$t_1 < T_0 \quad (11)$$

$$t_2 < T_3 \quad (11')$$

$$t_0 < T_1 \quad (12)$$

$$t_1 < T_4 \quad (13)$$

$$t_4 < T_1 \quad (14)$$

$$t_3 < T_2 \quad (15)$$

$$t_5 < T_4 \quad (16)$$

$$t_4 < T_5 \quad (17)$$

Constraints (9') and (10') are linked to constraints (9) and (10) by heat balances, and constraints (11') must be satisfied with  $T_3 > t_2$  in the problem itself. The variables are  $A_1, \dots, A_4, t_1, t_4, T_1, T_2, T_4, T_5$ , and  $J$ , but, having 9 equations and 11 variables, two independent variables can be chosen.

The solution procedure was constructed in the following manner choosing  $t_1$  and  $t_4$  as independent variables.

a. Rewrite expression (5) to (8) in explicit form for  $A_i = g_i(T, t)$ ,  $i = 1, \dots, 4$  and linearize to obtain

$$\begin{aligned} dA_1 &= (\partial g_1 / \partial t_1) dt_1 + (\partial g_1 / \partial T_1) dT_1 \\ dA_2 &= (\partial g_2 / \partial t_1) dt_1 + (\partial g_2 / \partial T_4) dT_4 \\ dA_3 &= (\partial g_3 / \partial t_4) dt_4 + (\partial g_3 / \partial T_1) dT_1 + (\partial g_3 / \partial T_2) dT_2 \\ dA_4 &= (\partial g_4 / \partial t_4) dt_4 + (\partial g_4 / \partial T_4) dT_4 + (\partial g_4 / \partial T_5) dT_5 \end{aligned} \quad (7.27)$$

b. Rewrite expression (1) to (4) in a similar form:

$$\begin{aligned} dT_1 &= -(wc_1 / WC_1) dt_1 \\ dT_4 &= (wc_1 / WC_2) dt_1 \\ dT_2 &= dT_1 - (wc_2 / WC_1) dt_4 \\ dT_5 &= dT_4 + (wc_2 / WC_2) dt_4 \end{aligned} \quad (7.28)$$

Then the SFG may be drawn as in Fig. 7.10.



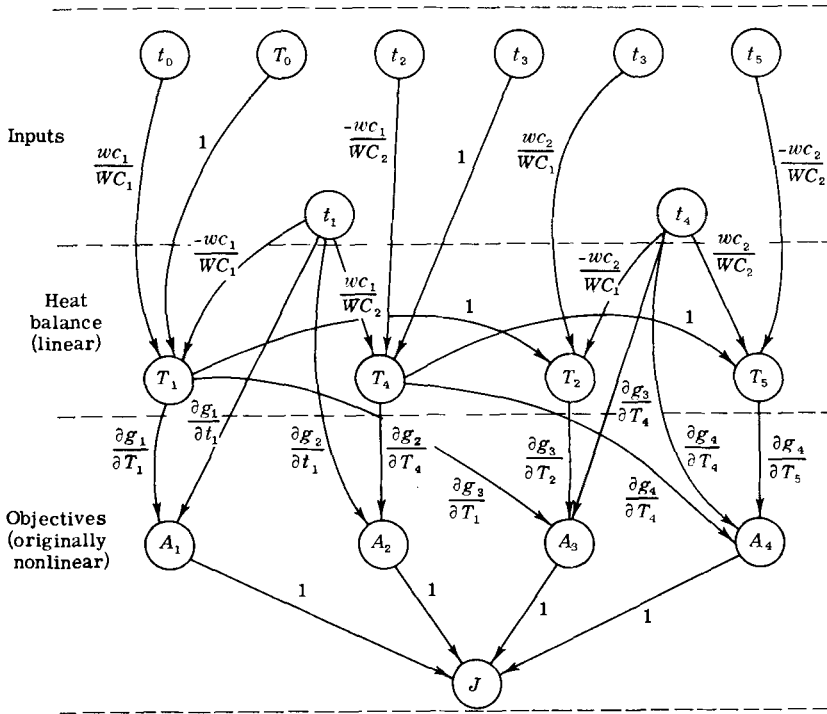


Fig. 7.10. SFG for four heat exchangers system. The  $dt$ 's,  $dT$ 's, and  $dA$ 's are replaced by  $A$ 's,  $t$ 's, and  $T$ 's. (This makes no difference because the SFG is used only for transmittance calculation.)

One of the fortunate situations is that, since the given heat balances are linear equations, the flow graph for the heat balances is exactly the same as for the  $dt$ 's and  $dT$ 's, except for the constant input nodes. As shown previously, the computation of gradient vectors (in terms of transmittances) does not depend on the constant input modes. Therefore the flow graph Fig. 7.10 serves both for calculating the gradient vector and for heat balance calculations. For instance:

$$T_2 = (T_{t_0 \rightarrow T_2})(t_0) + (T_{T_0 \rightarrow T_2})(T_0) + (T_{t_1 \rightarrow T_2})(t_1) \\ + (T_{t_3 \rightarrow T_2})(t_3) + (T_{t_4 \rightarrow T_2})(t_4) \quad (7.29)$$

$$\partial J / \partial t_1 = T_{t_1 \rightarrow J} \quad (7.30)$$

Also,  $\partial J / \partial T_i$ ,  $i = 1, 4, 2, 5$  is calculated as

$$\partial J / \partial T_2 = (T_{t_1 \rightarrow T_2})(\partial J / \partial t_1) + (T_{t_4 \rightarrow T_2})(\partial J / \partial t_4) \quad (7.31)$$

(Note that this  $\partial J/\partial T_2$  satisfies the heat balances but  $\partial J/\partial T_2 = T_{T_2 \rightarrow J}$  does not.)

Thus, at each iteration, the heat balances with given  $t_1$  and  $t_4$ , and then the  $A_i = g_i$  ( $t$ 's,  $T$ 's) are calculated, and finally the constrained gradient vectors  $(\partial J/\partial t_1, \partial J/\partial t_4, \partial J/\partial T_1, \partial J/\partial T_2, \partial J/\partial T_4, \partial J/\partial T_5)$  are obtained.

c. The temperature constraints are separated into two parts. First, we have the strict inequalities. If two temperatures, for instance  $t_1$  and  $T_0$  in expression (11) get infinitesimally close, then  $A_1$  goes to infinity. Therefore, if  $t_1$  is close enough to  $T_0$ ,  $-\partial J/\partial t_1$  is expected to have a negative sign and  $t_1$  can never be equal or greater than  $T_0$ . For this reason inequality constraints (11)–(17) are used only for obtaining the limits in the Fibonacci search.

On the other hand, constraints (9) and (10) seriously affect the solution. If  $t_1 < t_0$ ,  $A_1$  changes sign and  $A_1$  can go to minus infinity as  $t_1$  goes to minus infinity. Therefore, constraints (9) and (10) were applied for the LP part of the MFD.

This was done by rewriting the temperature constraints at the  $K$ th iteration;

$$\begin{aligned} t_0 - [t_1^K - (\partial J/\partial t_1)\lambda] &\leq 0 & \text{or} & & (\partial J/\partial t_1)\lambda &\leq t_1^K - t_0 \\ [t_1^K - (\partial J/\partial t_1)\lambda] - t_2 &\leq 0 & \text{or} & & -(\partial J/\partial t_1)\lambda &\leq t_2 - t_1^K \end{aligned} \quad (7.32)$$

and so on.

Equations (7.32) are of the form

$$p_i \lambda \leq q_i, \quad i = 1, \dots, 4$$

corresponding to temperature constraints (9) and (10). Also

$$p_i \lambda < q_i, \quad i = 5, \dots, 11$$

corresponding to temperature constraints (11), (12), (13), ... (17). If  $p_i$  is negative,  $\lambda$  is not bounded by constraint  $i$ .

We now obtain the maximum  $\lambda$ ,  $\lambda_{\max}$ . If  $\lambda$  is bounded by  $i = 1, 2, 3$ , or  $4$ , and the condition set forth in (e) is satisfied, we go to the LP, otherwise a Fibonacci search is used.

d. If we had followed the MFD exactly, the constraints of the LP problem would have to be changed at each iteration according to the boundaries on which the iteration point exist. Hence, in following the method of feasible direction, the LP problem is constructed as follows:

Maximize

$$Z = -(\partial J/\partial t_1)r_1 - (\partial J/\partial t_4)r_4 \quad (7.33)$$

Subject to

$$t_1^K + r_1 \geq t_0 \quad (21)$$

$$t_1^K + r_1 \leq t_2 \quad (22)$$

$$t_4^K + r_4 \geq t_3 \quad (23)$$

$$t_4^K + r_4 \leq t_5 \quad (24)$$

$$-1 \leq r_1 \leq 1, \quad -1 \leq r_4 \leq 1 \quad (25)$$

where superscript  $K$  shows the numbers of the iteration.

Four extra constraints (25) are introduced by normalization of the  $(r_1, r_4)$  vector; that is, the number of constraints has been doubled. This may cause problems when the number of independent variables is large. Fortunately, in this problem, the  $(r_1, r_4)$  vector is bounded by constraints (21)–(24), so these extra constraints could be omitted.

Additionally, in the larger problem with nine heat exchangers, the lower bound was eliminated by a change of variables. For instance, in constraint (21)

$$r_1 \geq t_0 - t_1^K$$

letting

$$R_1 = r_1 - (t_0 - t_1^K),$$

the constraint now is  $R_1 \geq 0$ .

e. It is necessary to decide whether the LP is to be used or not. Since it is impossible to know if the iteration point is exactly on the boundary when using a digital computer, the LP is used if the point is close enough to the boundary so that the distance to the boundary is less than one percent of the distance to the origin ( $t$ 's all zero) in the  $n$ -dimensional Euclidean space.

f. The logarithmic mean temperature causes trouble in the computer calculations. If  $\Delta t_1 = \Delta t_2$ , mathematically,

$$t_{lm} = (\Delta t_1 - \Delta t_2) / \ln(\Delta t_1 / \Delta t_2) = \Delta t_1$$

But the computer does not calculate  $t_{lm} = (0.) / \ln(1.)$ . Therefore, the computer program was constructed for the nine heat exchanger case so that the logarithmic means is replaced by an arithmetic mean, if

$$|(\Delta t_1 - \Delta t_2) / (\Delta t_1)| \leq 0.01.$$

g. Additionally, the Fibonacci search is used to obtain the  $\lambda$  which gives the optimum point in the direction given by the gradient vector or by the

LP solution for  $\mathbf{r}$  in Eq. (7.33):

$$\begin{aligned} t_1 &= t_1^K + r_1\lambda, & t_4 &= t_4^K + r_4\lambda \\ T_1 &= T_1^K + R_1\lambda, & T_2 &= T_2^K + R_2\lambda \\ T_4 &= T_4^K + R_4\lambda, & T_5 &= T_5^K + R_5\lambda \end{aligned}$$

where  $r_1$  and  $r_4$  are the solution to (7.33) or  $-\partial J/\partial t_1$  and  $-\partial J/\partial t_4$ , respectively, and  $R_1, \dots, R_5$  are obtained as described in part d.  $\lambda$  which gives minimum  $J$  is obtained by a Fibonacci search.

The calculation algorithm is briefly discussed in Appendix J. The computer program is available from the authors.

## Results and Discussion

The results for the four heat exchanger systems are shown in Table 7.3 and Figs. 7.11 and 7.12. The three cases corresponding to the hot stream inlet temperatures given in Fig. 7.9 were solved. This problem has two independent variables, so the trajectories along the iterations are nicely shown in two dimensions.

In cases I and II, the solution moves toward the boundary,  $t_2 = 100^\circ\text{F}$ , and on the boundary (actually very close to the boundary) the direction vector to move along is outside the feasible region. Then the LP solution corrects the direction and an optimum is found. Figure 7.12 shows the correction to the direction vector (as does Table 7.3).

The direction vector at point (1) of case I is  $(-0.4, -365.0)$  and slightly oriented to the direction of decreasing  $t_1$ , however,  $t_4$  cannot be decreased any more. Therefore the direction vector is changed to  $(-8, -0)$ , where  $(-0)$  is a very small negative number. The optimum solution to the linearized problem at point ① is now  $(t_1 = 100^\circ\text{F}, t_4 = 100^\circ\text{F})$ . But the problem is not linear and a Fibonacci search finds the minimum to be at point ②.

The direction vector at point ② is  $(0.01, -722)$  which makes an angle of about  $90^\circ$  measured from the  $t_4 = 100^\circ\text{F}$  line. Then, the direction is corrected to  $(46.1, 0)$  which directs us to the point  $(t_1 = 150^\circ\text{F}, t_2 = 100^\circ\text{F})$ . However, the Fibonacci search shows the optimum at nearly the same point  $(103.9, 100.0)$ , so this is taken to be the optimal solution.

In case III, the solution is not on the boundary and the trajectory for the gradient method is as shown in Fig. 7.11. In this case, the gradient method is not very effective. The conjugate gradient method may perhaps be more effective.

TABLE 7.3  
SOLUTION TO THE SYSTEM OF FOUR EXCHANGERS

Case	Iteration	$t_1$	$t_4$	$A_1$	$A_2$	$A_3$	$A_4$	$\Sigma A$ 's
I	0	120.0 3.0	120.0 -5.0	57.5	52.5	40.5	35.7	186.2
	1 <sup>a</sup>	108.0 -0.4 -8.0	100.0 -365.0 -0.0	18.1	70.2	0.0	62.2	150.5
	2	103.9 0.01 46.1	100.0 -722 -0.0	8.2	75.9	0.0	65.7	149.7
	3	103.0 0.005 46.1	100.0 -730 -0.0	8.2	75.9	0.0	65.7	149.7
	4	103.9	100.0	8.2	75.9	0.0	65.7	149.7
	0	120.0	120.0	57.5	81.1	40.5	69.3	248.5
	1	116.2	110.0	42.9	89.4	14.5	92.7	239.6
	2	120.3	107.1	58.8	80.4	10.2	88.3	237.7
	3	119.3	101.9	54.8	82.6	2.5	97.0	236.9
	4	119.5 0.3 30.5	100.1 -90 -0.1	55.6	82.1	0.1	99.0	236.8
II	5	120.5 -0.05 -20.5	100.1 -94 -0.0	59.8	79.9	0.0	96.9	236.7
	6	120.5	100.0	59.6	80.0	0.0	97.0	236.7
III	0	120.0	120.0	30.8	81.1	18.2	69.3	199.5
	1	131.4	130.7	56.1	53.9	37.6	38.3	185.9
	2	141.0	125.2	84.6	27.8	31.3	39.6	183.4
	3	138.3	116.9	75.7	35.6	17.6	52.2	181.0
	4	142.9	114.3	91.1	22.5	15.0	51.6	180.2
	5	142.0	109.9	88.0	25.1	9.7	57.1	179.8
	6	143.4	106.3	93.0	20.9	5.9	60.0	179.7
	7	143.4	106.2	93.1	20.9	5.8	59.9	179.7

<sup>a</sup> Second row shows the gradient (direction to move); third row shows the direction to move as corrected by LP.

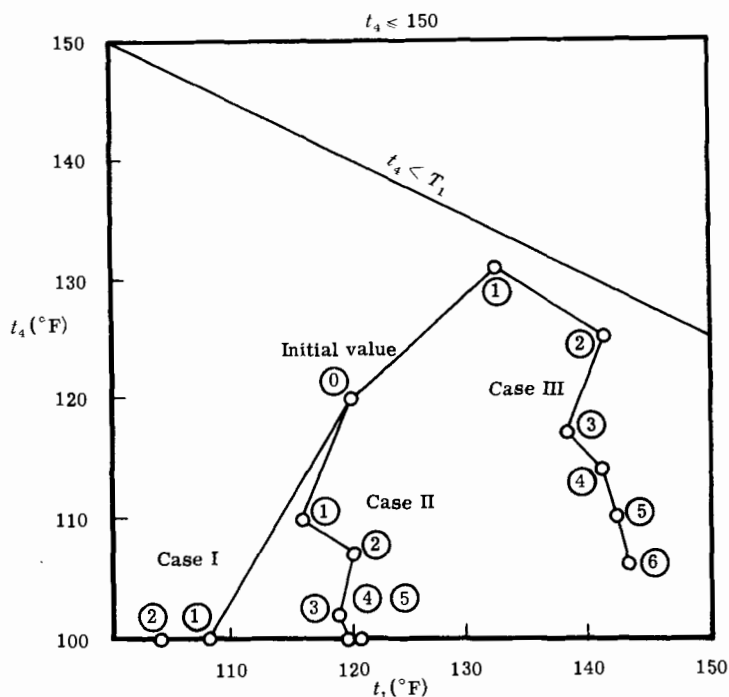


Fig. 7.11. Iterations in the solution procedure for the systems of four exchangers.

The results for the nine heat exchanger system shown in Figs. 7.8 and 7.3 are given in Tables 7.4 and 7.5.

Table 7.4 and Fig. 7.13 show the results of 25 iterations for initial condition 1. At iteration 2,  $HO(11)^{\dagger} = 180$ , which is on the boundary; then the LP solution corrected the direction so that  $HO(11) \geq 180$ . At iteration 5, the solutions,  $HO(11)$ — $HO(16)$  are near one of the extreme points and the  $\Sigma A$ 's jumped down to 71,885 sq. ft. from 83,250 sq. ft. This solution,  $\Sigma A$ 's = 71,885 sq. ft., is a far better optimum than Takamatsu's  $\Sigma A$ 's = 82,564 (for initial condition 2 in Table 7.5). Takamatsu's paper does not have any statement about the temperature constraints and of optimality criterion. Therefore it is impossible to find why his iterations terminated at 82,564.

$HO(11)$ ,  $HO(12)$ , and  $HO(13)$  left the boundaries in later iterations, while  $HO(14)$ ,  $HO(15)$ , and  $HO(16)$  never left the boundaries,  $HO(14)$

<sup>†</sup> See Fig. 7.8 for a designation of the  $HO$  variables.

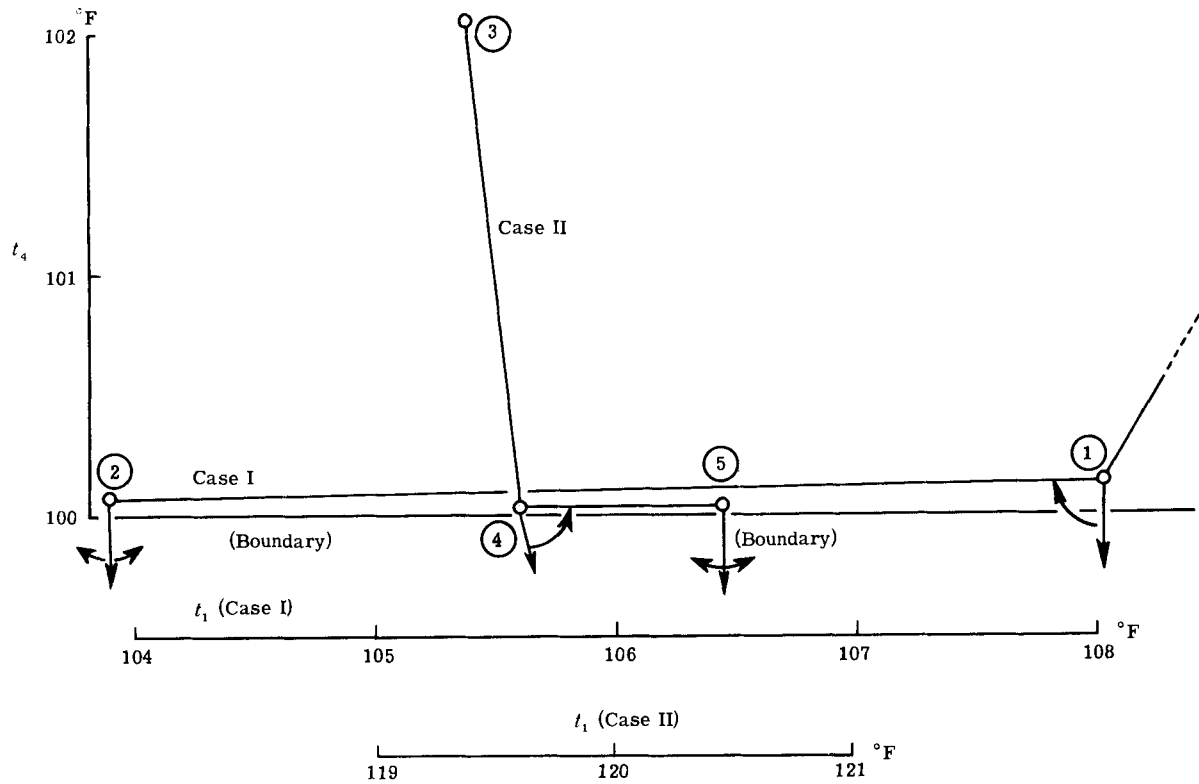


Fig. 7.12. Correction of direction by LP.

TABLE 7.4  
INITIAL VALUE 1 FOR NINE EXCHANGER SYSTEM

Iterations	HO(11)	HO(12)	HO(13)	HO(14)	HO(15)	HO(16)	$\Sigma A$
0	310.0	195.0	330.0	300.0	400.0	450.0	91,871
	-109.0	-67.3	-164.0	-142.0	-266.0	339.0	
1 <sup>a</sup>	304.8	191.8	322.2	306.7	387.3	466.1	86,113
	-74.9	-74.0	91.2	7.0	-41.7	-34.0	
2	292.9	180.0	336.7	307.9	380.7	460.7	84,258
	-18.6	-39.1	-16.9	-19.1	-102.0	134.0	
	-163.0	-0.0	-157.0	-158.0	-231.0	39.3	
3	283.8	180.0	328.0	299.1	367.9	462.9	83,383
	-85.3	-34.1	-73.2	-5.7	-40.1	-1.7	
	-154.0	-0.0	122.0	-149.0	-218.0	-313.0	
4	281.8	180.0	329.6	297.1	365.0	458.8	83,250
	-6.6	-30.4	36.4	3.4	-58.1	54.3	
	-152.0	-0.0	120.0	-147.0	-215.0	41.2	
5	130.1	180.0	450.0	150.1	150.1	500.0	71,885
	15.6	13.4	-17.8	146.0	-131.0	179.0	
	270.0	0.0	-270.0	350.0	350.0	0.0	
6	140.1	180.0	440.0	163.0	163.0	500.0	71,625
	17.3	10.2	34.5	109.0	-95.7	118.0	
	260.0	270.0	10.0	337.0	337.0	0.0	
7	171.0	212.0	441.2	203.9	203.9	500.0	70,825
	190.0	-4.8	18.1	79.0	-90.3	139.0	
	228.0	-32.8	8.8	-53.9	-53.9	0.0	
8	204.4	208.1	442.4	196.2	196.2	500.0	70,449
	0.9	-10.5	13.6	104.0	-95.7	144.0	
	196.0	-28.1	7.6	304.0	304.0	0.0	
9	215.8	206.4	442.9	213.9	213.9	500.0	70,359
	-0.3	-13.5	3.2	97.0	-98.2	158.0	
	85.8	-26.4	7.1	-63.9	-63.9	0.0	

(continued)



TABLE 7.4 (continued)

Iterations	HO(11)	HO(12)	HO(13)	HO(14)	HO(15)	HO(16)	$\Sigma A$
10	208.8	204.3	443.5	208.7	208.7	500.0	70,339
	3.1	-9.9	1.9	101.0	-101.0	159.0	
	191.0	-24.3	6.5	291.0	291.0	0.0	
11	212.1	203.9	443.6	213.7	213.7	500.0	70,332
	3.1	-10.8	-0.9	98.4	-101.0	163.0	
	188.0	-23.9	-264.0	-63.6	-63.6	0.0	
12	212.4	203.8	443.0	213.5	213.5	500.0	70,330
	2.8	-11.0	2.0	97.2	-99.6	159.0	
	188.0	-23.8	7.0	-63.5	-63.5	0.0	
13	217.5	203.2	443.2	211.8	211.8	500.0	70,317
	-1.1	-12.5	1.6	101.0	-100.0	160.0	
	-87.5	-23.2	6.8	288.0	288.0	0.0	
14	216.9	203.0	443.3	213.7	213.7	500.0	70,314
25	219.5	200.0	443.0	215.0	215.0	500.0	70,280

<sup>a</sup> Second row shows the gradient (direction to move); third row shows the direction to move as corrected by LP.

$= HO(15)$  and  $HO(16) = 500^\circ\text{F}$ . The solution  $\Sigma A$ 's are almost constant after iteration 10, while  $HO(11)$ — $HO(16)$  are still fluctuating.

Nine initial conditions were applied and the solutions at 25 iterations for each case are shown in Table 7.5. Although the values for the  $HO(I)$ 's are slightly different, the  $\Sigma A$ 's are about the same for each case. For practical purposes, the optimum solution is

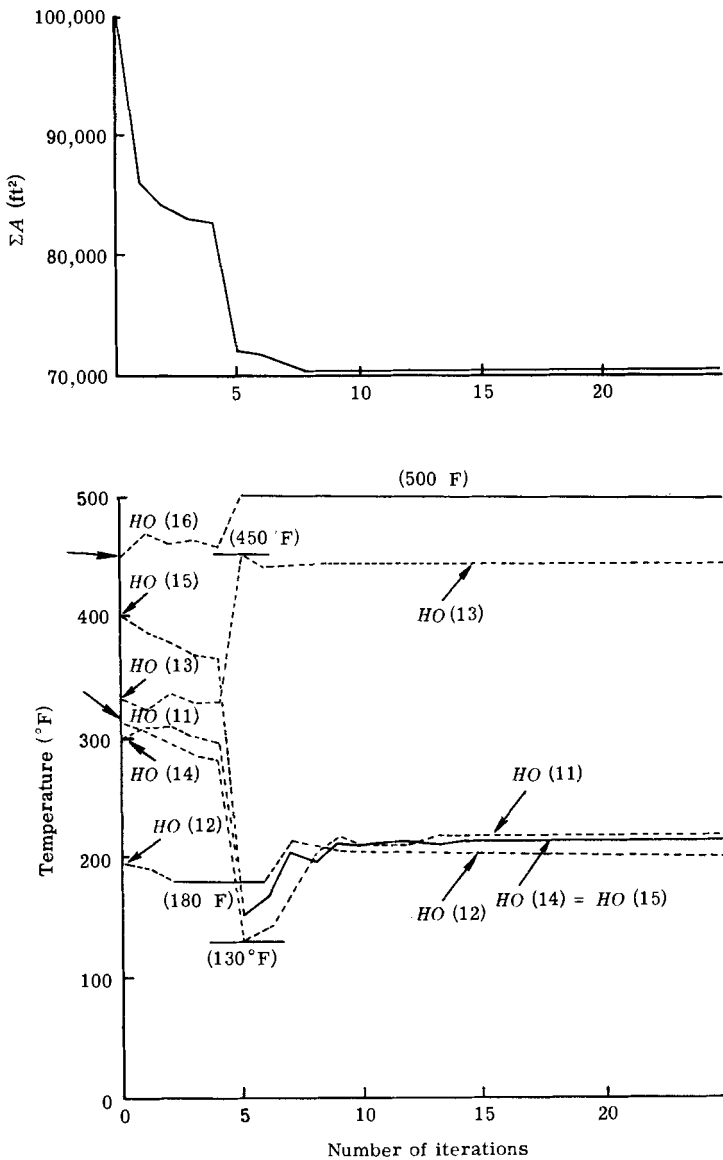
$$\begin{aligned} HO(11) &= 220^\circ\text{F}, & HO(12) &= 200^\circ\text{F}, & HO(13) &= 440^\circ\text{F} \\ HO(14) &= 215^\circ\text{F}, & HO(15) &= 215^\circ\text{F}, & HO(16) &= 500^\circ\text{F} \end{aligned}$$

where  $\Sigma A$ 's = 70,500 ft<sup>2</sup>.

Convergence toward the optimum point is very slow after the  $\Sigma A$ 's  $\cong 71,000$  sq. ft. In general, this is a difficulty with the gradient method and the conjugate gradient method may improve the iteration processes. Another problem which should be considered here is if there exists more than a single optimum.

TABLE 7.5  
SOLUTIONS TO THE NINE HEAT EXCHANGER SYSTEM AT ITERATION 25

Case	HO (11)	HO (12)	HO (13)	HO (14)	HO (15)	HO (16)	$A_1$	$A_2$	$A_3$	$A_4$	$A_5$	$A_6$	$A_7$	$A_8$	$A_9$	$\Sigma A$
1	220	200	443	215	215	500	4,930	988	3,869	19,762	1	457	26,137	14,130	5	70,280
2	215	199	443	212	212	500	4,582	929	3,666	19,904	0	435	26,350	14,398	4	70,269
3	220	199	443	213	213	500	4,955	933	3,717	19,797	0	464	26,300	14,095	5	70,266
4	212	199	439	217	220	499	4,374	914	4,160	19,442	100	742	26,159	14,567	276	70,737
5	221	194	441	216	216	498	4,972	692	3,921	19,926	0	568	25,881	14,012	468	70,441
6	217	198	443	213	213	500	4,676	857	3,712	19,947	0	445	26,311	14,301	0	70,250
7	223	188	440	217	217	498	5,009	353	3,952	20,058	0	687	25,805	13,882	784	70,529
8	213	201	443	210	210	500	4,425	1,058	3,592	19,754	0	436	26,469	14,560	0	70,297
9	221	185	439	217	220	500	4,778	235	4,006	20,061	98	757	26,488	14,039	1	70,463

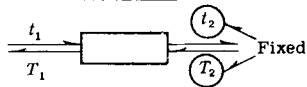
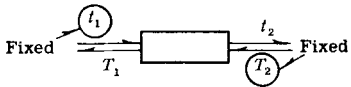


**Fig. 7.13.** Iteration procedure for nine-exchanger system. Solid line: boundary of feasible region (special case  $HO(14) = HO(15)$ ).

The theorems on optimality states that for the general programming problem of minimizing  $f(\mathbf{x})$  subject to  $g_i(\mathbf{x}) \leq 0$  ( $i = 1, \dots, m$ ) and  $\mathbf{x} \geq 0$ , if  $f(\mathbf{x})$  and all  $g_i(\mathbf{x})$  are convex functions, then a local minimum of  $f(\mathbf{x})$  subject to the constraint set is also a global minimum [5]. Applying this to the set of heat exchangers, since all the inequality constraints are linear, if all the equality constraints are convex function and if a minimum is found, then it is global minimum. Two cases where the constraint functions are treated as  $A = f(t's, T's)$ , are shown in Table 7.6.

In this problem,  $\beta = wc/WC < 1.0$ . Therefore, if the outlet temperature of the cold stream and the inlet temperature of the hot stream are fixed,

TABLE 7.6  
CONVEXITY OF HEAT TRANSFER FUNCTION<sup>a</sup>

$\beta(t_2 - t_1)$ $= T_2 - T_1$					
		$A(t_1)$	$A(T_1)$	$A(t_2)$	$A(T_1)$
$\beta > 1$	l. m.	Convex	Convex	$T_2 \uparrow t_2$ Concave $t_c$ Concave $t_1$	$T_2 \uparrow T_1$ Convex $t_1$ Concave $T_c$
	a. m.	$T_2 \uparrow t_1$ Convex $t_c$ Concave	$T_2 \uparrow T_1$ Convex $T_c$ Concave	$T_2 \uparrow t_2$ Concave $t_c$ Convex $t_1$	$T_2 \uparrow T_1$ Convex $t_1$ Convex $T_c$ Concave
$\beta = 1$	l. m.	Linear	Linear	$t_c = T_2 \uparrow t_2$ Concave $t_1$ Convex	$T_2 \uparrow T_1$ Convex $T_c = t_1$ Concave
	a. m.	Linear	Linear	$t_c = T_2 \uparrow t_2$ Concave $t_1$ Convex	$T_2 \uparrow T_1$ Convex $T_c = t_1$ Concave
$\beta < 1$	l. m.	Concave	Concave	$t_c \uparrow t_2$ Concave $T_2$ Convex $t_1$	$T_2 \uparrow T_1$ Convex $T_c$ Concave $t_1$
	a. m.	$t_c \uparrow t_1$ Convex $T_2$ Concave $t_2$	$T_c \uparrow T_1$ Convex $T_2$ Concave $t_2$	$t_c \uparrow t_2$ Concave $T_2$ Convex $t_1$	$T_2 \uparrow T_1$ Convex $T_c$ Convex $t_1$ Concave

<sup>a</sup> Note:  $wc(t_2 - t_1) = WC(T_2 - T_1)$ , then  $\beta = wc/WC$ ; l.m. = logarithmic mean temperature; a.m. = arithmetic mean temperature; subscript c denotes critical temperature between concave and convex which is a function of  $wc$ ,  $WC$ , and fixed temperature, but, the relative value to fixed temperature remains the same as shown by the diagram in each case.

the heat exchanger area for this particular heat exchanger is a concave function of the variable temperatures.

That is, heat exchanger areas 6, 8, and 9, at least, are concave function. Therefore, the existence of a single optimum to this problem cannot be proven. However, nine initial values, widely distributed in the feasible region, were tried and all converged to about the same place. Thus, the hypersurface of objective function in the area of convergence is probably very flat and slight local concavity may exist. The only way to find out if a global minimum exists is to calculate the complete contour (which is a hyper plane in six-dimensional space). This is practically impossible.

#### REFERENCES

1. T. Tonomura, Optimization by Signal Flow Graph Methods. M. S. Thesis, Univ. of Houston, Houston, Texas, December 1972.
2. F. S. Hillier and G. J. Lieberman, "Introduction to Operations Research." Holden-Day, San Francisco, California, 1967.
3. G. Zoutendijk, "Methods of Feasible Directions." Elsevier, Amsterdam, 1960; see also D. J. Wilde, *Ind. Eng. Chem.* **57**, 22 (1965).
4. T. Takamatsu, I. Hashimoto, and H. Ohno, Optimal design of a large complex system from the viewpoint of sensitivity analysis, *Ind. Eng. Chem. Process Des. Develop.* **9**, 368 (1970).
5. A. V. Fiacco and G. P. McCormick, The sequential unconstrained minimization technique for nonlinear programming. *Management Sci.* **10**, 360 (1964).

## CHAPTER 8

# RANKING BY FLOW-GRAPH METHODS

Mathematical puzzles and games have always been a challenge and inspiration to topologists. Indeed, as was pointed out in the preface, it was the famous Königsberg bridge puzzle which stimulated the Swiss mathematician Leonhard Euler to invent the graph-theoretic approach to route-finding puzzles. Another type of puzzle which has intrigued topologists for many years is that of tournaments and player rankings: Harary [1] credits the mathematician Redei with developing the first graph theory on tournaments in 1934.

Unfortunately, the available literature concerns itself exclusively with small, symmetric tournaments. In actual practice, the player (or team) ranking problem is much more complicated. Football team rankings, for example, involve an evaluation of hundreds of teams, and thousands of games in a very sparse matrix, since most of the teams do not play one another. A similar situation exists in college baseball, or basketball, tennis, and other sports. In general the polls by which rankings are usually established have a high credibility when it comes to selecting the top few teams, but they cannot be substantiated or justified below this. One of us (EJH) has had considerable experience with junior tennis rankings and can make

a very strong argument to the effect that the rankings made by traditional "by-guess by-God" methods are capricious and generally erroneous. The difficulty, of course, is that no human can assimilate all the information available. The best he can do is to look at the player's individual records. This is equivalent to looking at only a few games at a time and ignoring the information inherent in the results of the other thousands of games which have been played.

The key to obtaining equitable rankings is to formulate the problem in terms of a network and to look at all of the contests simultaneously. This, in essence, dictates a flow graph approach, and thus the theory, algorithms, and computer programs developed and applied to engineering systems in Chapters 1 through 7 should, in principle, be adaptable to ranking problems.

### Tennis Rankings

Figures 8.1, 8.2, and 8.3 represent the results of three tennis tournaments. Referring to Fig. 8.1 we see that player number two beat, successively, players 3, 4, 6, and then 13. Player number 13 beat players 12, 11, and 17 and lost to player number 2. Figures 8.2 and 8.3 have a similar interpretation. Figure 8.4 is an inventory of branches (matches) in the tournaments the  $a \rightarrow b$ 's in column two signify that player  $a$  beat player  $b$ .

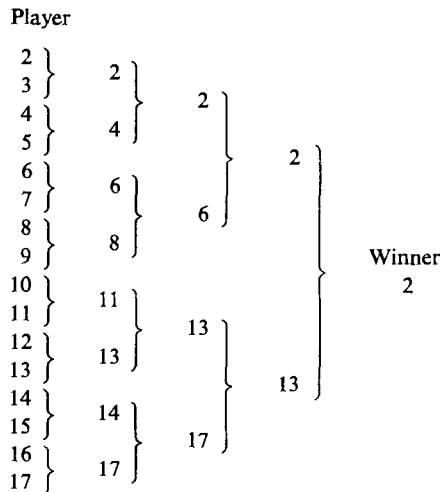
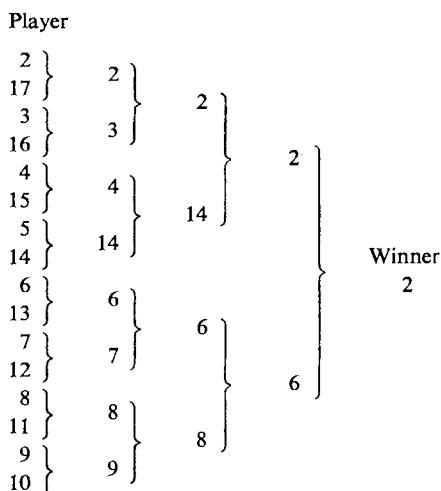
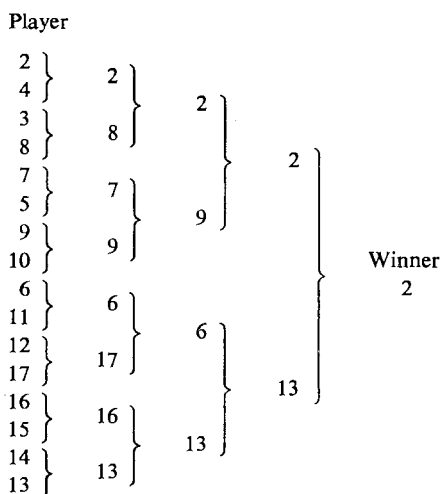


Fig. 8.1. Tournament I.



**Fig. 8.2.** Tournament II.

In assessing the ability of a person (or team) in any competition the three most important criteria should be won-and-loss records, the “quality” of the win, and the margin of win. A fighter, for example, who has a ten wins—one loss record against weak opponents may be a much poorer fighter than a man with a 10-4 record who fought strong opponents and had decisive wins. The key to achieving equitable rankings, therefore, rests with



**Fig. 8.3.** Tournament III.



our ability to evaluate a contestant's performance in terms of the environment in which he competes.

A ranking system based on the flowgraph principles and computer programs developed in this book has been designed and tested on the 1970 Texas Father-Son rankings and the 1971 national boys 16's tennis rankings.

Inventory of Branches	
Origin → End	Origin → End
2 → 3	8 → 9
2 → 4	8 → 11
2 → 6	8 → 9
2 → 13	9 → 10
2 → 17	9 → 10
2 → 3	13 → 17
2 → 14	13 → 14
2 → 6	13 → 16
2 → 4	14 → 15
2 → 8	17 → 12
2 → 9	17 → 14
2 → 13	17 → 16
3 → 16	16 → 15
4 → 5	14 → 4
4 → 15	14 → 5
6 → 7	13 → 6
6 → 8	13 → 11
6 → 13	13 → 12
6 → 7	11 → 10
6 → 8	9 → 7
6 → 11	8 → 3
6 → 17	7 → 5
7 → 12	

**Fig. 8.4.** Inventory of matches, computer input.

The method consists of simply enumerating all the paths and loops in the network. Consider for example, player number 17. From the inventory of branches, Fig. 8.4, we see that he is involved in the following path sequences:

17 → 12  
 17 → 14  
 17 → 16  
 17 → 14 → 15  
 17 → 14 → 4  
 17 → 14 → 5  
 17 → 16 → 15  
 17 → 14 → 4 → 5  
 17 → 14 → 4 → 15

TABLE 8.1  
TENNIS RANKINGS, COMPUTER OUTPUT

Player	Paths	Score	Average path length	Ranking
2	360	1362	3.79	1
13	76	248	3.26	2
6	76	215	2.82	3
8	17	35	2.06	4
17	9	17	1.88	5
9	5	7	1.40	6 (tie)
14	5	7	1.40	6 (tie)
3	2	3	1.50	7
4	2	2	1.00	8 (tie)
7	2	2	1.00	8 (tie)
11	1	1	1.00	9 (tie)
16	1	1	1.00	9 (tie)
5	0	0	0	10 (tie)
10	0	0	0	10 (tie)
12	0	0	0	10 (tie)
15	0	0	0	10 (tie)

As seen in Table 8.1, the computer output correctly shows that player 17 has accumulated nine paths and a total score of 17, (the branch gains are assigned a value of one and summed). The best player (number 2) has the most paths (360) and his paths are longer than those of inferior players since he beat people who have beaten other people. Thus his total score, 1,362, and average path length, 3.79, shows he should be ranked number one.

Players 3, 4, and 7, for example, each have two paths, but players 4 and 7 only beat people who had beaten no one else, whereas player 3 beat someone who had another victory. Since one of player 3's paths was longer, he has a higher score and is ranked above players 4 and 7. Players 4 and 7 must be ranked equally. If necessary or desirable, relative scores and/or weights can be used as branch gains. (In doing the national 16's rankings, for example, the committee decided to assign a double weight to the national championship (Kalamazoo) tournament.)

### Implementation Difficulties

There are two serious difficulties in implementing this ranking system. Consider tournament number one. The total number of matches played (which is always one less than the total number of players) is fifteen. When the same group of players participate in a second tournament, the winner

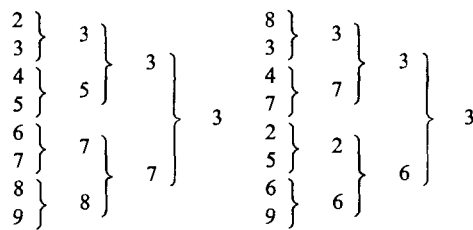
again picks up fifteen paths, and, in addition, he accumulates all of the beaten players' paths from the first tournament plus their wins in the second tournament. If there are  $n$  tournaments involving  $N$  players of roughly equal skill (i.e. each is capable of winning a tournament), the approximate upper limit on the possible number of paths is  $N \cdot (N/2)^n$ . For the system of 16 players and three tournaments just considered, the total possible paths for any player is about 1,000. It should be emphasized that this is only a crude estimate of the upper limit; the actual number of paths for the best player may be considerably less. However, in applications involving a large number of players and many tournaments, the number of paths becomes astronomical. For example, in the 1971 National Boys' Rankings there was a matrix of eighty-five players, seven tournaments and 285 matches. Using an IBM 370 Model 155 with 200K words of assigned core, and a maximum path length of 40,000, the run was terminated after twenty minutes. The 370 had in this time span been able to only compute 24 players and every one of these had more than 40,000 paths!

The second difficulty is one of program logic. A player who is fortunate enough to have one or two good wins will continue to accumulate paths even if he stops playing. Indeed, one good win can have a (possibly) anomalous effect on the entire structure.

Figure 8.5 demonstrates some of the possible "anomalies" which can develop. (The word anomaly is in quotes because these are the type of things ranking committees must deal with, and often there really is no one correct decision.) Columns one to six give the number of paths and score (path lengths) for tournaments I; I and II; and I, II, and III. Player three has won all of them and has the highest paths and score. All rankings are in logical order.

Columns 7 and 8 show the results of substituting tournament IIIA for III. Tournament IIIA was won by player 5 who has moved from next to last to second place in the process. This again is logical. Note also that player 4 has moved up to third place by virtue of his win over player 3, and that player 3, who won no matches at all in tournament III has a higher score (511 versus 427) than he did when he won the tournament; he has all the other players "working for him."

Columns 9 and 10 which show the results obtained using all four tournaments do pose a problem. We see that player 4 is in strong second place despite the fact that he won only a single match in four tournaments. It is difficult (though possible) to justify his being ranked ahead of player 7 who has six wins (including a win over number 4), or ahead of number 5, who has five wins.



INVENTORY OF PATHS. THESE PATHS EXIST BETWEEN NODES 1 AND 0

4

TABLE 8.3

INVENTORY OF PATHS. THESE PATHS EXIST BETWEEN NODES 1 AND 0<sup>a</sup>

ROW	DE_P	1	TRACE	2	3	4	5	6	7	8	9	GAIN	POWER	DE_S
30		1-->	3-->									0.0		0
31		1-->	3-->	4-->								0.0		1
32		1-->	3-->	5-->								0.0		1
33		1-->	3-->	6-->								0.0		1
34		1-->	3-->	7-->								0.0		1
35		1-->	3-->	7-->								0.0		1
36		1-->	3-->	7-->								0.0		1
37		1-->	3-->	8-->								0.0		1
38		1-->	3-->	9-->								0.0		1
39		1-->	3-->	5-->	6-->							0.0		2
40		1-->	3-->	5-->	8-->							0.0		2
41		1-->	3-->	5-->	9-->							0.0		2
42		1-->	3-->	6-->	9-->							0.0		2
43		1-->	3-->	5-->	6-->							0.0		3
44		1-->	3-->	7-->	8-->							0.0		2
45		1-->	3-->	7-->	8-->							0.0		2
46		1-->	3-->	7-->	8-->							0.0		2
47		1-->	3-->	7-->	9-->							0.0		2
48		1-->	3-->	7-->	8-->							0.0		2
49		1-->	3-->	7-->	8-->							0.0		2
50		1-->	3-->	8-->	9-->							0.0		2
51		1-->	3-->	5-->	8-->							0.0		3
52		1-->	3-->	7-->	8-->							0.0		3
53		1-->	3-->	7-->	8-->							0.0		3
54		1-->	3-->	7-->	8-->							0.0		3
55		1-->	3-->	7-->	8-->							0.0		3
56		1-->	3-->	7-->	8-->							0.0		3
57		1-->	3-->	7-->	8-->							0.0		3
58		1-->	3-->	9-->	4-->							0.0		2
59		1-->	3-->	5-->	9-->							0.0		3
60		1-->	3-->	6-->	9-->							0.0		3
61		1-->	3-->	5-->	6-->							0.0		4
62		1-->	3-->	8-->	9-->							0.0		3
63		1-->	3-->	5-->	8-->							0.0		4
64		1-->	3-->	7-->	8-->							0.0		4
65		1-->	3-->	7-->	8-->							0.0		4
66		1-->	3-->	7-->	8-->							0.0		4
67		1-->	3-->	7-->	8-->							0.0		4
68		1-->	3-->	7-->	8-->							0.0		4
69		1-->	3-->	7-->	8-->							0.0		4
70		1-->	3-->	9-->	2-->							0.0		2
71		1-->	3-->	5-->	9-->							0.0		3
72		1-->	3-->	6-->	9-->							0.0		3
73		1-->	3-->	5-->	6-->							0.0		4
74		1-->	3-->	8-->	9-->							0.0		3
75		1-->	3-->	5-->	8-->							0.0		4
76		1-->	3-->	7-->	8-->							0.0		4
77		1-->	3-->	7-->	8-->							0.0		4
78		1-->	3-->	7-->	8-->							0.0		4
79		1-->	3-->	7-->	8-->							0.0		4
80		1-->	3-->	7-->	8-->							0.0		4
		2-->	2-->									0.0		4

3

50

136

<sup>a</sup> The 1 represents a dummy node.

**A Solution to the Implementation Problem**

A rather ingenious way of solving both the path length and the “anomalous” ranking problem is to truncate the paths at approximately one tenth the longest path length. Inspection of Tables 8.2 and 8.3 a printout of the first fifty paths of players 3 and 4, reveals a very interesting phenomena. Recall that player 4 has won only one match, hence all but one of his paths is indirect, and involves three or more players ( $4 \rightarrow 3 \rightarrow x$ ). Player three, on the other hand, has nine victories and hence can branch out from nine places. Thus, in a truncated system, where all players have the same number of paths, the better players will have a *lower* total score by virtue of their direct wins over good players and the greater number of direct wins.

This is borne out by columns 11 and 12 of Fig. 8.5 which ranks the players logically—and consistently: 3-5-7-4-6-9-8-2. A comparison of won-loss records, Table 8.4 bears this out. Player 6 is correctly ranked ahead of player 9 because he had “better” wins. Whether or not player 4 is correctly ranked requires some human intervention. It may be that his win over number three was a fluke, in which case he should be ranked lower. This will have to be a “judgment call.”

The program was tested on the 1970 Texas Father and Son and 1971 National Boys’ tennis tournaments results. In the former case, which involved 77 players and 90 matches all paths were calculated. The Boys’

TABLE 8.4  
WON-LOSS RECORDS AND SCORE

Player	Total wins	Total losses	Total score
3	9	1	136
5	4	3	169
7	5	4	172
4	1	4	178
6	3	4	191
9	3	4	196
8	2	4	197
2	1	4	230

rankings involved 289 matches and 87 players and only 5,000 (of the more than 40,000) paths were used. Creditable results were obtained in both cases. Parenthetically, it should be pointed out that only about  $\frac{1}{4}$  of the match results were used because ranking committees were using summary won-loss records and we wanted to start from the same data base. This, in itself, is an indictment of the way ranking committees conventionally operate.

**REFERENCE**

1. F. Harary, "Graph Theory," p. 206. Addison-Wesley, Reading, Massachusetts, 1969.



This page intentionally left blank

## CHAPTER 9

# UNDIRECTED GRAPHS

### Undirected Graphs

An undirected graph is one in which the edges are “scalar” rather than “vector” in nature insofar as they have no direction. This does not imply that an edge must have magnitude; it is simply a line. It is possible, of course, to have graphs in which some edges are directed and others are not. Accordingly, much of the terminology and concepts of undirected graphs, has been applied (indiscriminatorily in many cases) to directed graphs. Conversely many of the definitions used in the analysis of signal flow graphs have been adopted by topologists concerned primarily with nondirected graphs. A brief table of equivalent nomenclature is

#### *Undirected graph (primarily)*

1. End vertex
2. Circuit
3. Tree
4. Chain progression (sequences)
5. Edge progression (sequences)
6. Closed chain (edge progression)

#### *Signal flow graph (primarily)*

1. Output or input node
2. Loop
3. Acyclic graph
4. Path
5. Path
6. Loop

### Trees

A fundamental concept in graph theory is that of a *tree*, which is defined as a connected subgraph of a graph which satisfies the following definitions and theorems<sup>†</sup>:

*Definition 1.* A tree contains no circuits.

*Definition 2.* A tree contains the vertices of the original graph, and thus *spans* it.

*Definition 3.* There exists only one path between any two vertices of a tree.

*Theorem 3A.* A tree containing  $v$  vertices has  $(v - 1)$  elements.

*Definition 4.* The *cotrees* or complements of a graph are the residual (connected and nonconnected) edges of the graph  $G$  from which the tree is formed. An element of the complement is called a *chord* or *link*.

*Theorem 4A.* There are  $(e - v + 1)$  chords in a connected graph of  $v$  chords and  $e$  edges.

*Definition 5.* A fundamental ( $f$ ) circuit of a connected graph is a unique circuit containing one chord (of the cotree) and the unique tree path formed by some or all of the branches of a tree between the two vertices of the chord.

*Theorem 5A.* There are exactly  $(e - v + 1)$   $f$ -circuits formed by a chord and its unique tree.

These concepts are now illustrated with reference to Fig. 9.1. Figure 9.1a shows the original graph. Three of the eight possible tree subgraphs are represented by (b), (c), and (d). There exist three more trees similar to (b) and two more analogous each to (c) and (d).

Applying the tree theorems and definitions to the graph of Fig. 9.1a and the tree (b), we note that:

1. The tree (b) contains no circuits (Definition 1).
2. The tree (b) contains all the  $v$  vertices of the original graph (Definition 2); has only one path between adjacent nodes (Definition 3); and contains  $(v - 1)$  elements (Theorem 3A).

<sup>†</sup> Proofs of the theorems in this chapter can be found in most of the books listed in the bibliography at the end of the book.

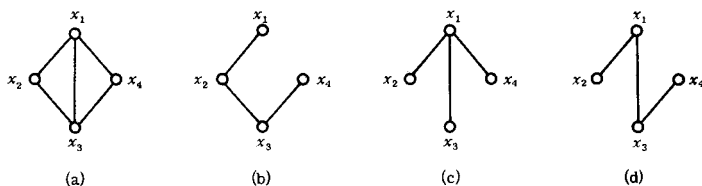


Fig. 9.1. (a)–(d) An example of trees. (From S. Seshu and M. B. Reed, "Linear Graphs and Electrical Networks," p. 25. Addison-Wesley, Reading, Massachusetts, 1961.)

3. Two cotrees (chords)  $x_1 - x_3$  and  $x_1 - x_4$  exist (Definition 4). This is in accord with Theorem 4A which states that there should be  $(e - v + 1) = (4 - 3 + 1) = 2$  chords.

4. Applying either chords  $x_1 - x_3$  or  $x_1 - x_4$  to the tree forms a fundamental ( $f$ ) circuit (Definition 5). There should be exactly  $(e - v + 1) = 2$   $f$ -circuits (Theorem 5A).

There are many important applications of tree concepts to engineering; the determination of Kirchoff circuit equations, network analysis, design synthesis and output variable selection, to name a few. All of these, unfortunately, necessitate more theorems and definitions. Before introducing these, we provide a diversion by presenting two elementary applications of tree structures to decision processes (logic trees and branch and bound). An advanced treatment of these topics is given in Chapter 11.

## Logic Trees

*Logic trees* can be used for enumerating possible combinations of events, situations, or decisions. An example due to Bellman serves to illustrate the use of tree structures in logic situations (see Bellman *et al.* [1]).<sup>†</sup> Given an urn with three balls numbered 1, 2, and 3, how many different ways can the balls be drawn out one at a time?

At the first draw (the *root of the tree*) we may draw number 1, 2, or 3. The possibilities are depicted graphically in Fig. 9.2a.

At the second draw we obtain either of the remaining balls, the possibility being as shown in Fig. 9.2b.

On the third draw we exhaust all possibilities.

<sup>†</sup> Logic tree examples of this type now appear in most high school elementary analysis texts.

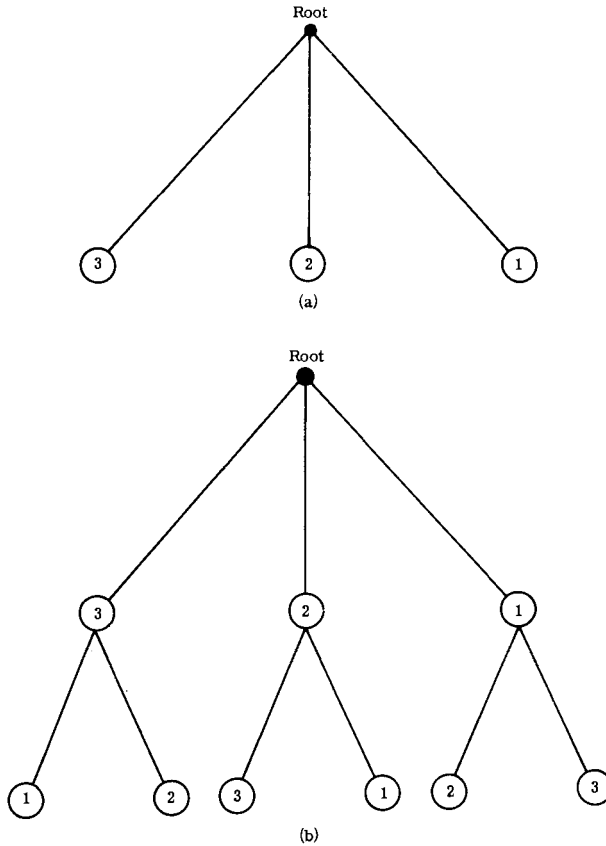


Fig. 9.2. (a)–(b) Logic trees.

In a structure such as Fig. 9.2c, each path from start to end represents one logical possibility, one possible order of events. The six possible orders are

2, 1, 3	1, 3, 2	3, 2, 1
1, 2, 3	3, 1, 2	2, 3, 1

Logic trees are useful route finders, providing loops are prohibited, or nonexistent. Consider the flow chart of Fig. 9.3 where the edge numbers are distances, and the nodes locations. We wish to go from 0 to 8 by the shortest possible path without retracing (“revisiting”) any of the nodes.

We begin at node 0, from where we can proceed to 1 or 2. The tree begins as illustrated in Fig. 9.4(a).

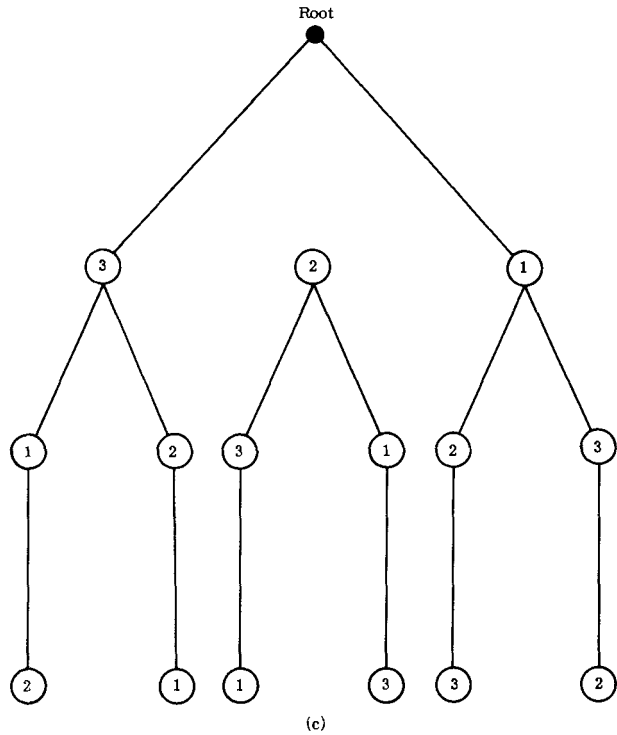


Fig. 9.2. (c) Logic tree.

From 1 we can go to 3, and from 2, to 3 or 4 [Fig. 9.4(b)].  
From 3 we can proceed to 1, 2, or 5, and from 4, to 2, 5, 6. The routes 0-1-3-1, 0-2-4-2 and 0-2-3-2 are illegal loops and are not shown [Fig. 9.4(c)].  
The completed diagram of Fig. 9.4(d) is often called a *forest* since it contains many trees.  
Each path from 0 to 8 along a solid line represents a possible route. There are eight possibilities. The sum of the edge gains give the distances. Thus, along, 0-2-4-6-7-8, the sum of edge gains is  $(c + f + i + k + j)$ . One can

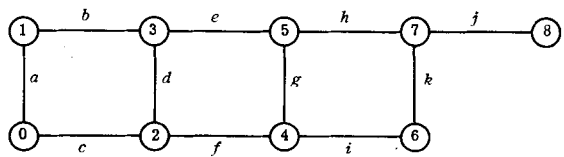


Fig. 9.3. A route.

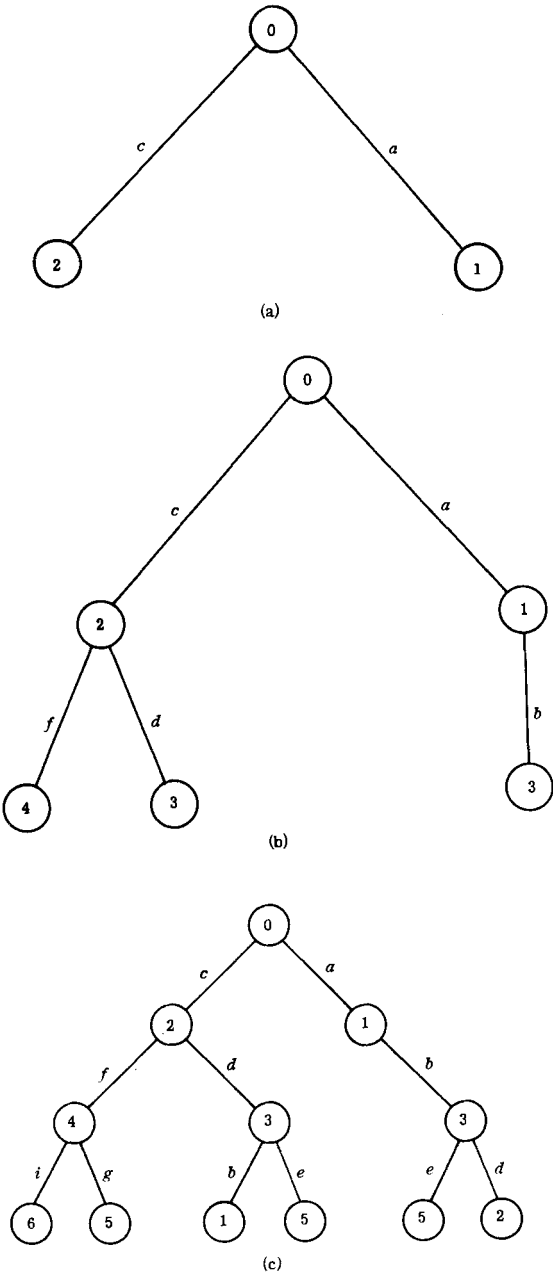


Fig. 9.4. (a)–(c) Development of tree structure.

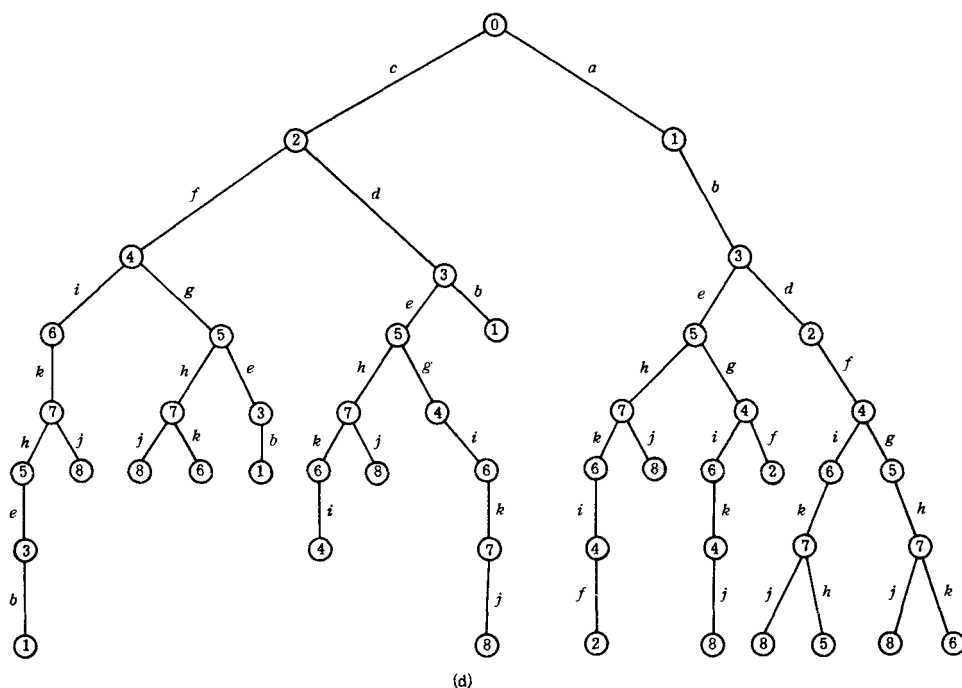


Fig. 9.4. (d) Complete forest.

thus pick the best (shortest) route from 0 to 8 by comparing the sum of the gains of the eight possible routes.

The application of decision trees to process design and synthesis is currently being explored at a number of universities. Graph theory is being used in conjunction with combinatorial mathematics to formulate systematized dos and don'ts in how to combine processing units, chemicals and equipment.

### Branch and Bound Methods

Branch and bound and related methods have been applied successfully to engineering design [2]. A much referenced, basic article on the method is that of Lawler and Wood [3]. The technique will now be illustrated by an "assignment problem," which could arise when one wants, for example, to optimally assign streams to heat exchangers. A generalization of the branch and bound and related methods is developed in Chapter 11.



The tableau of Fig. 9.5 is a typical assignment matrix where  $A, B, C, D$  are the streams to be assigned to units 1, 2, 3, 4, the elements being cost of the assignment [4]. The problem is to find an optimum assignment of the streams to minimize cost. There are  $4! = 24$  possible assignments.

We notice first that there are "tight" upper and lower optimal bounds to the assignments. The lower bound gives as a value of the objective function  $P = (11 + 1 + 8 + 21) = 41$ . This is not a feasible solution, since it assigns stream  $D$  to units 1 and 4, and we assume multiple assignments to be unfeasible. The upper bound,  $P = 94 + 88 + 88 + 82$ , is also infeasible. Upper and lower bounds are also present after an assignment has been made. If, for example, stream  $A$  is assigned to unit 1, the lower bound is  $P = 94 + (10 + 8 + 21) = 133$  which, coincidentally, is one of the 24 feasible (though not optimal) solutions to the problem.

		Units			
		1	2	3	4
Assignee	$A$	94	1	54	68
	$B$	74	10	88	82
	$C$	62	88	8	76
	$D$	11	74	81	21

Fig. 9.5. Assignment table.

The idea of the branch-and-bound method (where a minimum is to be found) is to use the value of the best feasible solution found thus far as an upper bound, and then establish feasible subsets which are inspected to see if their lower bounds exceed the upper bound of the previously found feasible solution. Those subsets whose lower bounds exceed the previous upper bound are discarded, and further subsets are generated. The process is repeated until a feasible solution is found such that the previous feasible upper bound is smaller than the new lower bound. When this occurs, the optimum has been reached.

Figure 9.6 shows the structure for the branch and bound tree. After the first assignment, the upper bound, feasible solution gives 104 for the value of the objective function. Since the *best possible assignment* of variables beginning with  $A - 1$  and  $C - 1$  produces  $P$  which is greater than the best feasible solution obtained (133 and 138 as against 104), we eliminate these trees from the forest.

We now branch out of the  $P = 88$  tree, forming the three subsets shown by successively allocating streams  $A, B$ , and  $C$  to unit 2. Our upper feasible

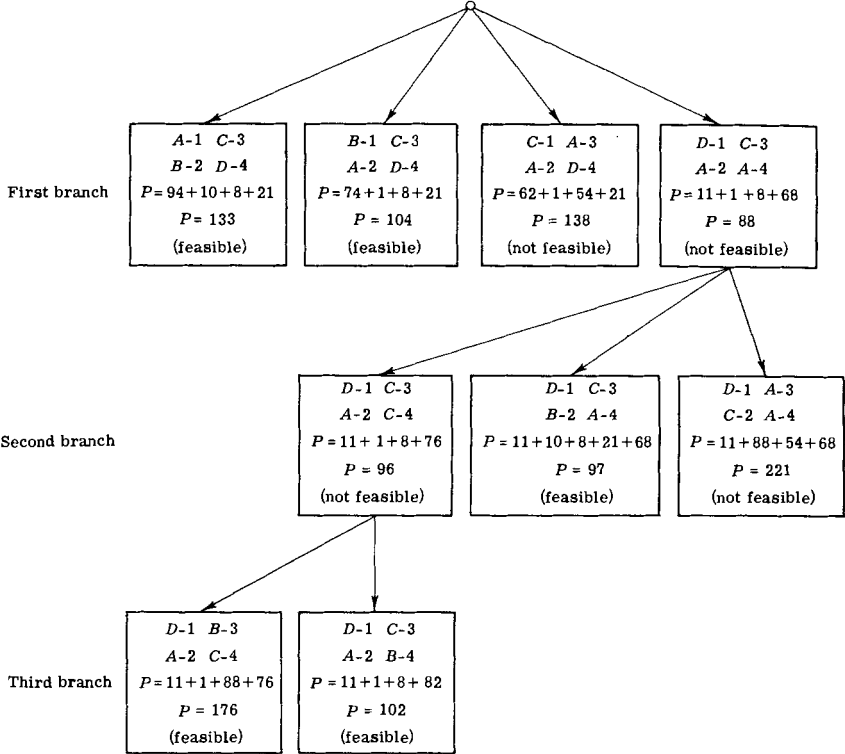


Fig. 9.6. A branch and bound tree.

bound now is  $P = 97$ ; and we branch out of the tree  $P = 96$ , assigning streams  $B$  and  $C$ , successively, to unit 3. The two (feasible) solutions found have lower bounds larger than the previous feasible bound thus the optimal solution was the assignment  $D-1, B-2, C-3, A-4$ .

The branch and bound method enabled us to find the optimum solution in 9 attempts. This is an appreciable saving in computation time; the alternative being to evaluate the  $4!$  possible combinations.

Trees—Further Theorems and Definitions

A forest was previously defined as a collection of trees.

*Theorem.* A forest having  $k$  trees and  $v$  vertices has  $(v - k)$  edges.

This is illustrated in Fig. 9.7 where  $k = 3, v = 10$ , and there are  $(v - k) = (10 - 3)$  edges.

*Definition.* The *rank* of a graph is  $(v - k)$ .

*Definition.* The *circuit rank*,  $\mu$ , (*cyclomatic number*, *nullity*, *connectivity*, *first Betti number*) is the minimum number of edges which must be removed to reduce a cyclic structure to an acyclic (tree) structure.

*Theorem.* The circuit rank of a graph is  $\mu = (e - v + k)$ .

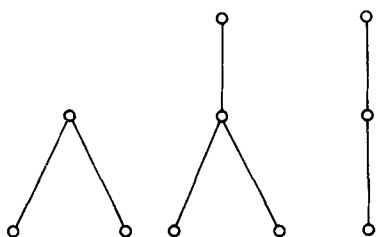


Fig. 9.7. A forest of three trees.

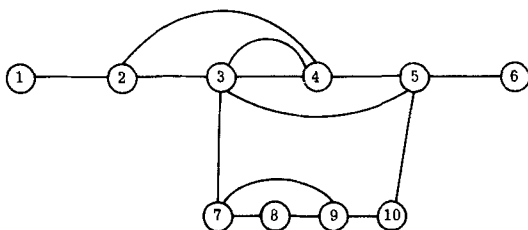


Fig. 9.8. Flow graph of a chemical process.

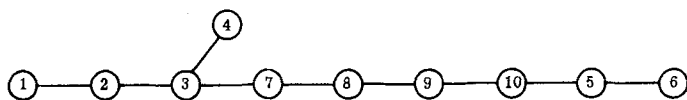


Fig. 9.9. Figure 9.8 rendered acyclic.

For example, the cyclic graph in Fig. 9.8 is a representation of a chemical plant previously considered in Chapter 6 (Fig. 6.14). To render this graph acyclic, we must remove (tear) a minimum of  $\mu = 14 - 10 + 1 = 5$  edges. Removing edges 2-4, 3-4, 4-5, 3-5, and 7-9, results in the acyclic tree shown in Fig. 9.9. The “tears” are not unique.

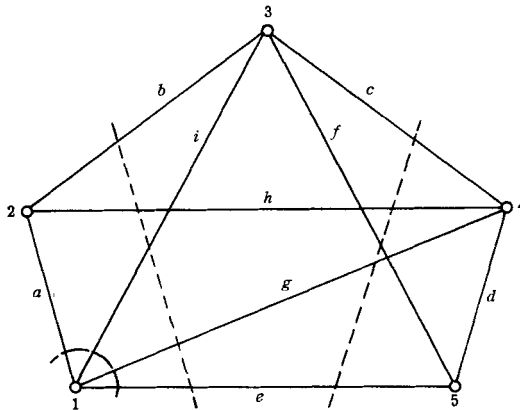
**Disconnecting Sets and Cut-Sets**

A *cut-set* is a set of edges of a connected graph which, when removed from  $G$  reduces its rank by one, providing that no subset of the removed edges has the same property. Removal of the cut-set of edges always divides the graph into two parts.

The rank of a graph is  $v - k$ , hence removal of a cut-set of edges from a connected graph reduces its rank by one and usually yields another graph which has only one vertex. As an example of cut-sets, we have the edges *aige*, *chgfe*, and *bhige* in Fig. 9.10.

For this figure cut-sets can be obtained by inspection, but this is not always so. Some graphs must be redrawn to identify the cut-sets.

*Theorem.* If a graph is a tree, there will be exactly  $(v - 1)$  branches and consequently  $(v - 1)$  *fundamental cut-sets*.



**Fig. 9.10.** Examples of cut-sets. (From S. Seshu and M. B. Reed, "Linear Graphs and Electrical Networks," p. 28. Addison-Wesley, Reading, Massachusetts, 1961.)

Application of cut-sets generally involve matrix manipulations and further discussions will be referred until after the section on matrix representation of graphs. Before proceeding with this topic, we introduce another special type of graph of importance in engineering.

**Bipartite Graphs**

A graph is bipartite if its vertices can be separated into two disjoint sets  $f_i$  and  $v_i$  in such a way that every edge has one end point in  $f_i$  and the other

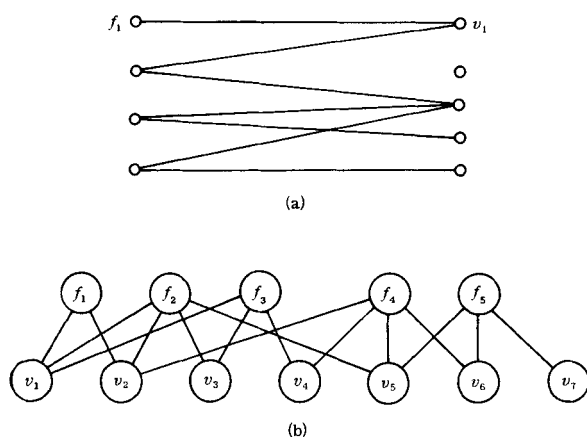


Fig. 9.11. (a) A bipartite graph. (b) A bipartite graph of Eq. (9.1).

in  $v_i$ . A bipartite graph is generally displayed by collecting the vertices  $f_i$  and  $v_i$  into separate columns or rows (Fig. 9.11).

Graphs of this type can usefully portray the structure of sets of equations such as [5]

$$\begin{aligned}
 f_1(v_1, v_2) &= 0, & f_2(v_1, v_2, v_3, v_5) &= 0 \\
 f_3(v_1, v_3, v_4) &= 0, & f_4(v_2, v_4, v_5, v_6) &= 0 \\
 f_5(v_5, v_6, v_7) &= 0
 \end{aligned}
 \tag{9.1}$$

which becomes Fig. 9.11(a).

The equations (which need not be linear) form the  $f$  nodes, the variables the  $v$  nodes. The number of edges associated with a node defines the local degrees of freedom. For the  $f_1$  node, the local degree is 2, for the  $v_5$  node it is 3.

As has been shown by Lee *et al.*, bipartite graphs have certain properties which are useful in flow sheet analysis, in developing equation solving algorithms, and in the selection of design variables.

The design problem, from a most general standpoint, consists of establishing the degrees of freedom in the system and then choosing the design variables which are to be fixed; the remaining variables, which must be calculated, form the "output set." The total number of variables which must be specified to solve the equations is the degrees of freedom  $F$  which is the excess of variables  $M$  over equations  $N$ .

$$F = M - N \tag{9.2}$$

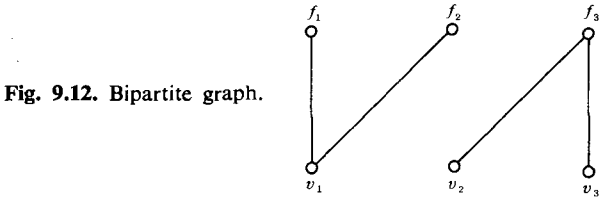


Fig. 9.12. Bipartite graph.

For the set of Eqs. (9.2),  $F = 7 - 5 = 2$ . Thus the problem is to assign values for two variables in such a manner that each of the five remaining variables is assigned in some optimum way to the five equations. [Assignment problems constitute the major use of bipartite graphs. See, for example Harary *et al.* [6] for a discussion of the “picnic” problem and the “marriage” problem.]

The set-theoretic theorem of Hall [7] is useful in the analysis of bipartite graphs. This theorem gives the condition that there must be at least  $K$  unspecified variables associated with each and every group of  $K$  equations ( $K = 1, 2, \dots, N$ ) for an output assignment of only one variable to one equation to be possible.

For example the bipartite graph in Fig. 9.12 describes an insolvable set of equations because, for  $K = 2$ , only one variable is assigned to equations  $f_1$  and  $f_2$ .

### Assigning Output Sets

We now assign a direction to the edges of a bipartite graph, to identify which variable is solved from which equation. An arrow from  $f_i$  to  $v_i$  indicates that equation  $f_i$  is solved for  $v_i$ . The node  $v_i$  then becomes an input to all  $f$ 's (other than  $i$ ) to which it is connected. The design variables form an input set and can be segregated.

Figure 9.13 shows the graph belonging to Eq. (9.1) where  $v_2$  and  $v_3$  are

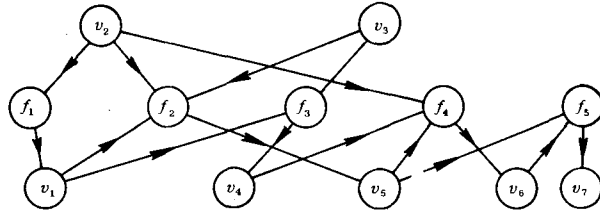


Fig. 9.13. Bipartite graph of Eq. (9.1).

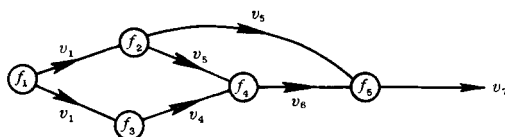


Fig. 9.14. Bipartite graph of Eq. (9.1).

chosen as design variables and the output variables  $v_1$ ,  $v_4$ ,  $v_5$ ,  $v_6$ , and  $v_7$  are obtained respectively from  $f_1$ ,  $f_3$ ,  $f_2$ ,  $f_4$ , and  $f_5$ .

A somewhat neater arrangement of Fig. 9.13 is shown in Fig. 9.14. This graph shows the precedence order of the system of equations. Starting from the left and proceeding to the right the set of five equations can be solved one at a time. There is no need for the simultaneous solution of any of the five equations. An alternative way of recognizing this is to note that there are no loops; the structure is acyclic.

Lee *et al.* in their interesting paper, extend these ideas to the development of algorithms for the selection of design variables and process flow sheet orderings which minimize loop formation and maximize the ease with which output variables can be computed. The identification of recycle loops and output sets in chemical process flow sheets constitutes one of the major applications of graphs and graph theory. There is a rich literature in this field, much of which is summarized by Ledet and Himmelblau [8].

## REFERENCES

1. R. Bellman, K. L. Cooke, and J. A. Lockett, "Algorithms, Graphs, and Computers," p. 7. Academic Press, New York, 1970.
2. K. F. Lee, A. H. Masso, and D. F. Rudd, *Ind. Eng. Chem. Fundam.* **9**, 48 (1970).
3. E. L. Lawler and D. E. Wood, *Oper. Res.* **14**, 699 (1966).
4. F. S. Hillier and G. J. Lieberman, "Introduction to Operations Research," p. 566. Holden-Day, San Francisco, California, 1967.
5. W. Lee, J. Christensen, and D. Rudd, *AIChE J.* **12**, 1104 (1966).
6. F. Harary, R. L. Norman, and D. Cartwright, "Structural Models," p. 384. Wiley, New York, 1965.
7. P. Hall, *J. London Math. Soc.* **10**, 26 (1935); see also Harary *et al.* [6].
8. W. P. Ledet and D. M. Himmelblau, *Advan. Chem. Eng.* **8**, 185 (1970).

## CHAPTER 10

# MATRIX REPRESENTATION OF GRAPHS

Matrix representations of directed and undirected graphs have both esthetic and utilitarian value. It is possible to deduce incidence relations, circuits, and cut-sets by matrix theorems and manipulations. Since most matrix manipulations involve either Boolean or Modulo 2 algebra, the reader unfamiliar with these topics is referred to Appendix D for a review.

Graph-matrix terminology is at best varied. A summary of the nomenclature adopted here appears in Table 10.1. We now offer examples of each of the matrix representations listed in this table.



TABLE 10.1  
TABLE OF MATRIX DEFINITIONS

---

<i>Description of elements</i>	
<i>A. Nondirected graphs</i>	
1. Incidence (vertex)	$a_{ij} = 1$ if edge $i$ is connected to vertex $j$ . $a_{ij} = 0$ if edge $i$ is not connected to vertex $j$ .
2. Circuit matrix	$a_{ij} = 1$ if the edge $j$ is in circuit $i$ . $a_{ij} = 0$ if the edge from $i$ to $j$ is not in circuit $i$ .
3. Cut-set matrix	$a_{ij} = 1$ if edge $j$ is in cut-set $i$ . $a_{ij} = 0$ if edge $j$ is not in cut-set $i$ .
<i>B. Digraphs</i>	
4. Adjacency (associated, relation structural, Boolean)	$a_{ij} = 1$ if there is an edge from vertex $i$ to vertex $j$ . $a_{ij} = 0$ if there is no edge from $i$ to $j$ .
5. Transition (Markov)	$a_{ij} = p_{ij}$ where $p$ is the probability of going from vertex $i$ to vertex $j$ .
6. Incidence (vertex)	$a_{ij} = 1$ if edge $j$ is directed out of vertex $i$ . $a_{ij} = -1$ if edge $j$ is directed into vertex $i$ . $a_{ij} = 0$ if there is no connection between vertex $i$ and edge $j$ .
7. Circuit	$a_{ij} = 1$ if the edge $j$ is in circuit $i$ and has circuit orientation. $a_{ij} = -1$ if the edge $j$ is in circuit $i$ and is opposite to circuit orientation. $a_{ij} = 0$ if the edge is not in the circuit.
8. Cut-set matrix	$a_{ij} = 1$ if edge $j$ is in cut-set $i$ and is circuit oriented. $a_{ij} = -1$ if edge $j$ is in the cut-set $i$ and opposite to the circuit. $a_{ij} = 0$ if edge $j$ is not in cut-set $i$ .
9. Connection matrix	$a_{ij}$ = branch gain from vertex $i$ to vertex $j$ (as illustrated in Fig. 2.9).

---

# Nondirected Graphs

## 1. INCIDENCE MATRIX $\mathbf{I}_N$

The graph connections are shown by a matrix having  $v$  rows and  $e$  columns. With reference to Fig. 10.1:

$$\mathbf{I}_N = \begin{array}{c} \text{Vertex} \\ \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{matrix} \end{array} \begin{array}{c} \text{Edge} \\ \begin{matrix} a & b & c & d & e & f & g \end{matrix} \end{array} \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (10.1)$$

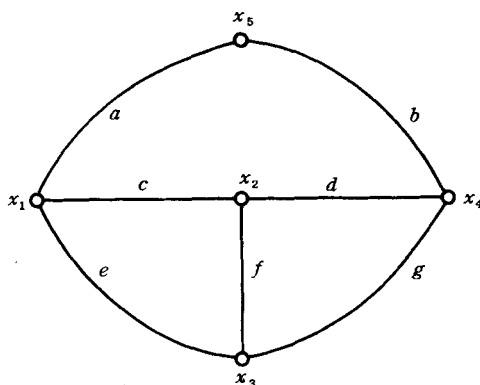


Fig. 10.1. A nondirected graph.

There are several theorems of interest with respect to incidence matrices. Two of the most useful are:

*Theorem.* Every column of  $\mathbf{I}_N$  contains two entries.

*Theorem.* The rank of  $\mathbf{I}_N$  is  $(v - 1)$ .

## 2. CIRCUIT MATRIX $\mathbf{C}_N$

There are four loops in Fig. 10.1: 1 ( $a-b-c-d$ ), 2 ( $c-e-f$ ), 3 ( $f-d-g$ ), 4 ( $a-b-g-e$ ). The circuit matrix is

				Edge						
	Circuit	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>		

$$\mathbf{C}_N = \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad (10.2)$$

Several theorems of interest apply

*Theorem.*  $\mathbf{I}_N \mathbf{C}_N^T = 0$  and  $\mathbf{I}_N^T \mathbf{C}_N = 0$ , in field modulo 2 algebra.

This permits us to verify if all the loops in a given flow graph have been identified. Since

$$\mathbf{I}_N \mathbf{C}_N^T = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (10.3)$$

there are indeed only four loops in the structure shown in Fig. 10.1.

*Theorem.* The rank of the circuit matrix is  $(e - v + 1)$ .

### 3. CUT-SET MATRIX, $\mathbf{S}_N$

There are four fundamental cut-sets in Fig. 10.1:

1 (*a-b*), 2 (*b-c-f-g*), 3 (*e-f-g*), 4 (*b-d-g*). The cut-set matrix is

				Edge						
	cut-set	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>		

$$\mathbf{S}_N = \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \quad (10.4)$$

Some applicable theorems of interest are

*Theorem.* The rank of  $S_N$  is  $(v - 1)$ .

*Theorem.* In modulo 2,  $C_N S_N^T = 0$ .

Thus

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (10.5)$$

This is a convenient way of testing whether the cut-sets have been appropriately identified.

## Digraphs

### 4. ADJACENCY MATRIX, $R$

The pertinent theorems will be discussed in terms of an application offered by Himmelblau [1]. Figure 10.2 shows a cyclic graph and its corresponding adjacency matrix  $R$ .

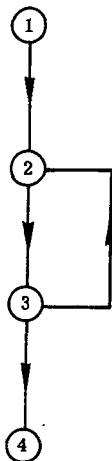


Fig. 10.2. A flow graph.

$$\begin{array}{c}
 \text{To node} \\
 \text{From node} \quad 1 \quad 2 \quad 3 \quad 4 \\
 \mathbf{R} = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}
 \end{array} \quad (10.6)$$

In the Boolean matrix the entries  $a_{ij}$  represent the one-step paths from vertex  $i$  to vertex  $j$ . These matrices have the interesting property that the  $n$ th power of  $\mathbf{R}$ ,  $\mathbf{R}^n$  shows the  $n$ -step paths in the flow graph. Thus  $\mathbf{R}^2$  contains the two step paths between (1-2-3) (2-3-2), etc. and  $\mathbf{R}^3$  the three step paths (1-2-3-2), (1-2-3-4), etc. If a graph is acyclic  $\mathbf{R}^N$  will be zero, when  $N$  is one more than the longest path in the graph.

$$\begin{array}{c}
 \begin{array}{c} 1 \quad 2 \quad 3 \quad 4 \\ \mathbf{R}^2 = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \end{array} \quad \begin{array}{c} 1 \quad 2 \quad 3 \quad 4 \\ \mathbf{R}^3 = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}
 \end{array} \quad (10.7)$$

If the graph contains loops, the adjacency matrix can be used to identify them. To accomplish this, first define the *reachability matrix*  $\mathbf{R}_r$ , which is the Boolean sum of all powers of  $\mathbf{R}$ . As the powers of  $\mathbf{R}$  become higher, one eventually reaches a point where  $\mathbf{R}_r$  no longer changes, since all  $n$ -step paths have been found and we are simply traversing the same loops. For example, after  $n = 3$ ,  $\mathbf{R}_r = \mathbf{R} + \mathbf{R}^2 + \mathbf{R}^3$ , thus

$$\begin{array}{c}
 \begin{array}{c} 1 \quad 2 \quad 3 \quad 4 \\ \mathbf{R}_r = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}
 \end{array} \quad (10.8)$$

If we were to form  $\mathbf{R}_r$  from  $\sum^4 \mathbf{R}^n$ , it would be a matrix identical to (10.8).

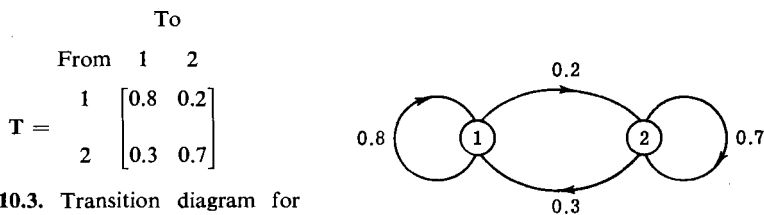
A useful property of the reachability matrix and its transpose,  $\mathbf{R}_r^T$  is that  $\mathbf{R}_r \mathbf{Q} \mathbf{R}_r^T$ , the elementwise Boolean multiplication, shows in the  $i$ th row all those vertices that are in the same cyclic loop with vertex  $i$ . For Fig. 10.2

$$\mathbf{R}_r \mathbf{Q} \mathbf{R}_r^T = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix} \quad (10.9)$$

Thus the loop in Fig. 10.2 is neatly exposed. The reader should, however, be made aware that in the case of nested loops precautions must be taken because the results are obscured.

### 5. TRANSITION MATRIX, $\mathbf{T}$

The transition matrix is basically an adjacency matrix with branch gains as elements. A primary use of these matrices is in the analysis of multistep transition probabilities for Markovian processes. Figure 10.3 shows a transition matrix corresponding to the given flow graph [2].



**Fig. 10.3.** Transition diagram for Markov problem.

States 1 and 2 correspond to, let us say, switch positions in a circuit subject to a random disturbance. After the first disturbance, if the switch is in position one, there is an 80% chance the switch returns to position one, and a 20% chance it will go to position 2. During the same (first) disturbance there is a 70% chance it will stay in state 2 (if it was there originally) and a 30% chance it will go to 1. The probability of the switch being in positions 1 or 2 depends on its original positions and the number of disturbances.

The second power of the transition matrix in Fig. 10.3 is

$$\mathbf{T}^2 = \begin{bmatrix} 0.7 & 0.3 \\ 0.45 & 0.55 \end{bmatrix} \quad (10.10)$$

0.7, for example, is the probability that the switch will return to position 1 after the second disturbance (providing it was in state 1 originally). The  $n$ th powers of  $\mathbf{T}$  gives the corresponding probabilities after the  $n$ th disturbance.

Connection and transition matrices are defined identically, the only difference being that the branch gains in transition matrices are generally probabilities.

## 6. INCIDENCE MATRIX, $I_D$

The next three matrices, the incidence, circuit, and cut-set matrices for digraphs will be introduced by means of an example taken from Smith *et al.* [3]. Their terminology and notation has been adopted to permit the reader to familiarize himself with flowgraph representations of other authors. We will also take this opportunity to acquaint the reader with the methods and language of systems analysis.

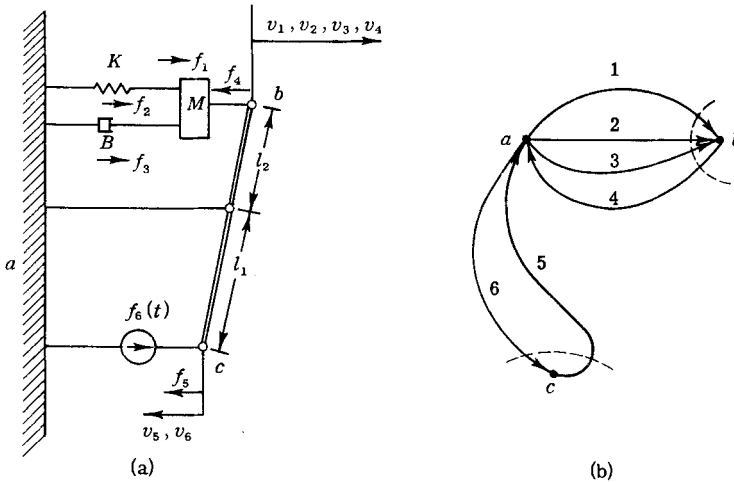


Fig. 10.4. (a) Mechanical system with (b) digraphs, tree, and cut-sets.

In the graph shown in Fig. 10.4, the vertices  $a$ ,  $b$ , and  $c$  represent the forces at the points indicated

$$f_a = f_4 + f_5 - f_6 - f_1 - f_2 - f_3 \quad (10.11)$$

$$f_b = f_1 + f_2 + f_3 - f_4 \quad (10.12)$$

$$f_c = f_6 - f_5 \quad (10.13)$$

In systems language these vertices are called *terminals* in the sense that they correspond to interfaces of components in the system. The variables

associated with these vertices are called *terminal variables*, and a tree sub-graph of Fig. 10.4 is referred to as a *terminal graph*. The expressions relating the forces to the velocity and system parameters are

$$dv_1/dt = f_1/M \quad (10.14)$$

$$df_2/dt = Kv_2 \quad (10.15)$$

$$f_3 = Bv_3 \quad (10.16)$$

$$f_4 = -(l_1/l_2)f_5 \quad (10.17)$$

$$v_5 = (l_1/l_2)v_4 \quad (10.18)$$

In matrix notation, combining (10.14), (10.15) and (10.16), (10.18), we obtain

$$\frac{d}{dt} \begin{bmatrix} v_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} 1/M & 0 \\ 0 & K \end{bmatrix} \begin{bmatrix} f_1 \\ v_2 \end{bmatrix} \quad (10.19)$$

$$\begin{bmatrix} f_3 \\ f_4 \\ v_5 \end{bmatrix} = \begin{bmatrix} B & 0 & 0 \\ 0 & 0 & -l_1/l_2 \\ 0 & l_1/l_2 & 0 \end{bmatrix} \begin{bmatrix} v_3 \\ v_4 \\ f_5 \end{bmatrix} \quad (10.20)$$

The  $M$ ,  $K$ , and  $B$  represent mass, spring constant, and damper, and  $f_6$  is the driving force. Following the nomenclature of Koenig *et al.* [4], velocity, (like voltage, pressure, temperature, and concentration) is designed an *across variable* and force (like current, torque, or heat) is a *through variable*. Velocity and force represent a set of *complementary variables*. Pressure and flow, and voltage and current, are two other examples of complementary variables.

Pertinent to the analysis of a dynamic system is the concept of a *state model* which establishes the general properties of a model as they relate to the system structure. A fundamental postulate of systems analysis is that the pertinent (behavioral) characteristics of each  $n$ -terminal component in an identified system structure are completely specified by a set of  $n - 1$  equations in  $n - 1$  pairs of oriented complementary variables identified by an arbitrarily chosen terminal graph. This suggests that we can obtain a *state equation* in terms of  $m$  variables (*the state vector*) which completely specify the remaining across and through variables by the following procedure: (a) we arbitrarily choose a *terminal graph* (tree) of the signal flow graph; (b) we would expect then, by proper manipulations, to produce a set of  $n - 1$  equations in  $n - 1$  pairs of variables which describe the system. In the case of Fig. 10.4 we have  $n = 3$ , so we anticipate two-state equations in two unknowns.



When we form a tree, there will be certain variables associated with it, and others with the cotree. It is customary to designate the through variables associated with the chords in the cotree and the across variables in the branches of the tree as *primary variables*. The across variables in the chords and the through variables in the branches are *secondary variables*. We will see that these functional relationships come about as a natural consequence of matrix partitioning. Indeed, these relationships can be shown to be axiomatic. Thus if we form a tree with branches 1 and 5, we would expect to get a relationship between  $(f_1, f_5)$  and  $(f_2, f_3, f_4, f_6)$  and another between  $(v_1, v_5)$  and  $(v_2, v_3, v_4, v_6)$ .

Referring to Table 10.1, the incidence matrix for the graph of Fig. 10.4 is

$$\mathbf{I}_D = \begin{array}{c} \text{Vertex} \\ \begin{matrix} a \\ b \\ c \end{matrix} \end{array} \begin{array}{c} \text{Edge} \\ \begin{matrix} 1 & 5 & 2 & 3 & 4 & 6 \end{matrix} \end{array} \left[ \begin{array}{cccccc} 1 & -1 & \vdots & 1 & 1 & -1 \\ -1 & 0 & \vdots & -1 & -1 & 1 \\ 0 & 1 & \vdots & 0 & 0 & 0 \end{array} \right] = \left[ \begin{array}{c|c} \mathbf{I}_B' & \vdots \\ \hline & \mathbf{I}_C' \end{array} \right] \quad (10.21)$$

We have ordered the matrix so that the elements corresponding to the tree are to the left, as indicated by the submatrix (dashed line).

In a directed incidence matrix the sum of the columns must equal zero (one edge in, one edge out). The rank of the matrix is  $(v - 1)$  so no information is lost, and the possibility of singularities can be reduced if we use only two rows of this matrix in subsequent manipulations.

## 7. CIRCUIT MATRIX, $\mathbf{C}_D$

Rather than writing the full circuit matrix, let us demonstrate its construction by entering only the fundamental circuits formed by adding chords to the tree formed by edges 5 and 1. There are  $(e - v + 1) = (6 - 3 + 1) = 4$  such circuits.

$$\mathbf{C}_D = \begin{array}{c} \text{Circuit} \\ \begin{matrix} (1, 2) \\ (1, 3) \\ (1, 4) \\ (5, 6) \end{matrix} \end{array} \begin{array}{c} \text{Edge} \\ \begin{matrix} 1 & 5 & 2 & 3 & 4 & 6 \end{matrix} \end{array} \left[ \begin{array}{cccccc} -1 & 0 & \vdots & 1 & 0 & 0 \\ -1 & 0 & \vdots & 0 & 1 & 0 \\ 1 & 0 & \vdots & 0 & 0 & 1 \\ 0 & 1 & \vdots & 0 & 0 & 0 \end{array} \right] = \left[ \begin{array}{c|c} \mathbf{C}' & \vdots \\ \hline & \mathbf{I} \end{array} \right] \quad (10.22)$$

The form of this matrix will always be as indicated on the right, where  $\mathbf{I}$  is the identity matrix.

### 8. CUT-SET MATRIX, $\mathbf{S}_D$

Choosing as cut-sets, (1, 2, 3, 4) and (5, 6), the form of the cut-set matrix is

$$\begin{array}{c} \text{Cut-set} \\ \mathbf{S}_D = \begin{array}{c} (1, 2, 3, 4) \\ (5, 6) \end{array} \begin{bmatrix} 1 & 0 & 1 & 1 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix} = \left[ \mathbf{I} \begin{array}{c} \vdots \\ \vdots \end{array} \mathbf{S}' \right] \end{array} \quad (10.23)$$

We now examine some of the necessary relationships among the circuit, cut-set, and incidence matrices and their application to the analysis of the system described by the flowchart.

a. For any circuit row vector  $\mathbf{c}$ , and any arbitrary vector  $\mathbf{x}$  of across variables identified by the  $e$  edges of a systems graph

$$\mathbf{c}\mathbf{x} = 0$$

Referring to Fig. (10.4) and matrix (10.22), we see that the so-called *fundamental circuit equations* for the graph are

$$\begin{bmatrix} -1 & 0 & 1 & 1 & -1 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_5 \\ v_2 \\ v_3 \\ v_4 \\ v_6 \end{bmatrix} = 0 \quad (10.24)$$

or

$$\left[ \mathbf{C}' \begin{array}{c} \vdots \\ \vdots \end{array} \mathbf{I} \right] \begin{bmatrix} \mathbf{V}_B \\ \mathbf{V}_C \end{bmatrix} = 0 \quad (10.25)$$

where  $\mathbf{C}'$  was defined by Eq. (10.22) and  $\mathbf{V}_B$  and  $\mathbf{V}_C$  refer to the branch and chord velocities. A few matrix manipulations will show that

$$\mathbf{V}_C = -\mathbf{C}'\mathbf{V}_B \quad (10.26)$$

thus

$$\begin{bmatrix} v_2 \\ v_3 \\ v_4 \\ v_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_5 \end{bmatrix} \quad (10.27)$$

This is the relationship between the branch and chord velocities.

b. For any cut-set vector  $\mathbf{s}$ , and any arbitrary vector  $\mathbf{y}$  of through variables identified by the edges  $e$  of the system graph:

$$\mathbf{sY} = 0 \quad (10.28)$$

Referring to Fig. 10.4 and matrix (10.23), the *fundamental cut-set equations* become

$$\begin{bmatrix} 1 & 0 & 1 & 1 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_5 \\ f_2 \\ f_3 \\ f_4 \\ f_6 \end{bmatrix} = 0 \quad (10.29)$$

or

$$\begin{bmatrix} \mathbf{I} & \vdots & \mathbf{S}' \end{bmatrix} \begin{bmatrix} \mathbf{F}_B \\ \mathbf{F}_C \end{bmatrix} = 0$$

Here  $\mathbf{S}'$  was obtained from Eq. (10.23) and  $\mathbf{F}_B$  and  $\mathbf{F}_C$  are the branch and chord forces.

From (10.29) it is relatively simple to show that

$$\mathbf{F}_B = -\mathbf{S}'\mathbf{F}_C \quad (10.30)$$

Thus, we obtain a relationship between the tree and cotree forces.

$$\begin{bmatrix} f_1 \\ f_5 \end{bmatrix} = \begin{bmatrix} -1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f_2 \\ f_3 \\ f_4 \\ f_6 \end{bmatrix} \quad (10.31)$$

c. Another axiom of systems theory is that each incidence vector  $\mathbf{i}$  must satisfy the *incidence equations*,

$$\mathbf{iY} = 0,$$

where  $\mathbf{Y}$  are the through variables. In accordance with our previous observations regarding the rank of  $\mathbf{I}_D$ , row  $a$  will not be used. This has the additional advantage of producing a square submatrix of branch gains, as indicated by the dashed partitions

$$\begin{bmatrix} -1 & 0 & \vdots & -1 & -1 & -1 & 0 \\ 0 & 1 & \vdots & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_5 \\ \hline f_2 \\ f_3 \\ f_4 \\ f_6 \end{bmatrix} = 0 \quad (10.32)$$

d. There are certain orthogonality relations between the circuit, cut-set, and incidence matrixes which permit us to verify the results of a, b, and c (or to obtain them by manipulation).

$$1. \quad \mathbf{C}^T + \mathbf{S}'' = 0 \quad (10.33)$$

verifying from Eqs. (10.22) and (10.23)

$$\begin{bmatrix} -1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} = 0$$

Thus the cut-set matrix can easily be deduced from the circuit matrix (or vice versa).

$$2. \quad \mathbf{S}' = [\mathbf{I}_B']^{-1}[\mathbf{I}_C''] \quad (10.34)$$

where  $[\mathbf{I}_B']^{-1}$  is the inverse of the branch portion of the incidence matrix and  $\mathbf{I}_C''$  is the chord submatrix [see Eq. (10.21)]. We can verify Eq. (10.34) using Eq. (10.21) and Eq. (10.23).

$$\begin{bmatrix} 1 & 1 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} -1 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

This gives the correct result. Note that only two rows of the incidence matrix were used. Had we used three, no inverse could have been defined, and the matrix multiplication would have been nonconformable.

All of the equations necessary for the synthesis of the state equation have been derived, and verified. We need now only make a few substitutions to arrive at the state equation for the system. The objective in these substitutions is to take Eq. (10.19) and replace the right-hand side with a function of  $v_1, f_2$  and the driving force,  $f_6$ .

Substituting for  $f_1$  and  $v_2$  from Eqs. (10.27) and (10.31),

$$\frac{d}{dt} \begin{bmatrix} v_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} 0 & -1/M \\ K & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ f_2 \end{bmatrix} + \begin{bmatrix} -1/M & 1/M \\ 0 & 0 \end{bmatrix} \begin{bmatrix} f_3 \\ f_4 \end{bmatrix} \quad (10.35)$$

Next we use Eq. (10.20) and substituting for  $v_3$ ,  $v_4$ , and  $f_5$ , from (10.27) and (10.31)

$$\begin{bmatrix} f_3 \\ f_4 \end{bmatrix} = \begin{bmatrix} B & 0 \\ 0 & -l_2/l_1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_6 \end{bmatrix} \quad (10.36)$$

Combining (10.36) and (10.35):

$$\frac{d}{dt} \begin{bmatrix} v_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} 0 & 1/M \\ K & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ f_2 \end{bmatrix} + \begin{bmatrix} -l/M & l/M \\ 0 & 0 \end{bmatrix} \begin{bmatrix} B & 0 \\ 0 & -l_1/l_2 \end{bmatrix} \begin{bmatrix} v_1 \\ f_6 \end{bmatrix} \quad (10.37)$$

Simplifying:

$$\frac{d}{dt} \begin{bmatrix} v_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} -B/M & -1/M \\ K & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ f_2 \end{bmatrix} + \begin{bmatrix} -l_1/Ml_2 \\ 0 \end{bmatrix} f_6 \quad (10.38)$$

Equation (10.38) is the state equation and the vector consisting of  $v_1$  and  $f_2$  is the state vector.

#### REFERENCES

1. D. M. Himmelblau, *Chem. Eng. Sci.* **21**, 425 (1966).
2. R. Howard, "Dynamic Probabilistic Systems," p. 9. Wiley, New York, 1971.
3. C. L. Smith, R. W. Pike, and P. W. Murill, "Formulation and Optimization of Mathematical Models," p. 415. Int. Textbook C., Lancaster, Pennsylvania, 1970.
4. H. E. Koenig, V. Tokada, and H. K. Kesavian, "Analysis of Discrete Physical Systems," p. 7. McGraw-Hill, New York, 1967.

## CHAPTER 11

### BRANCH AND BOUND, SEARCH TREE METHODS<sup>†</sup>

Branch and bound, introduced in Chapter 9 in the form of an assignment problem, uses the concepts of trees, logic trees, and bounds to solve combinatorial problems. The method is a powerful alternative to exhaustive enumeration on a computer, since the time and storage requirements for exhaustive enumeration increase exponentially with the number of variables, and, even large and fast machines can handle only very small problems. The name “branch and bound” comes from the particular approach used by Little, *et al.* [1] in their attempt to solve the famous operations research problem of the traveling salesman. But one finds a great variety of other labels, among which are “branch and exclude,” “branch search,” “branch and prune,” “bound and scan,” “implicit enumeration,” etc.

Branch and bound methods use “search trees,” each node of which represents a class of possible solutions to the problem. The union of all the pending nodes (hanging nodes not yet processed) represents the class of all possible solutions. The algorithm begins by assigning this last class

<sup>†</sup> Principal author: Jean-Paul Barthes, Chemical Engineering Department, Stanford University, Stanford, California.

to the root of the tree. Branching is used to replace one pending node by a set of pending nodes, the solution classes of which partition the solution class of the replaced node. A cost is computed for each pending node. This corresponds to a lower bound on the cost of any solution in the solution class of the node. If a node is known to contain no feasible solutions, the cost bound is infinite. The cost bounds induce an ordering of desirability on the pending nodes, which determines the branching in subsequent steps. The algorithm stops when it is not possible to generate any new node, or when a feasible solution with associated cost less than the lower bounds of the pending nodes has been found. See also [22a, 27a].

### Formal Definition of the Branch and Bound Technique

Let  $S = \{\sigma_j\}$  be the set of possible solutions to a problem  $\mathcal{P}$  of interest. Let  $|S|$ , the cardinality of  $S$ , be a finite number, and  $f$  be a function defined on the elements  $\sigma_j$  of subsets of  $S$ . We would like to find the solution  $\sigma^* \in S$  which minimizes the function  $f$ , and is *feasible*, i.e., satisfies a set of conditions  $\{C\}$ .

Suppose the problem has a property which allows us to make a partition  $\Pi$  of a subset  $S_{0lm\dots p}^{(i-1)}$  of  $S$ .

$$\Pi = \{S_{0lm\dots p1}^{(i)}, S_{0lm\dots p2}^{(i)}, \dots, S_{0lm\dots pq}^{(i)}\}, \quad q > 1 \quad (11.1)$$

where the subsets are defined recursively by

$$S_{0lm\dots pk}^{(i)} \neq \phi, \quad k = 1, 2, \dots, q \quad (11.2)$$

$$\bigcup_{k=1}^q S_{0lm\dots pk}^{(i)} = S_{0lm\dots p}^{(i-1)}, \quad (11.3)$$

$$S_{0lm\dots pk}^{(i)} \cap S_{0lm\dots pj}^{(i)} = \phi, \quad k, j = 1, 2, \dots, q, \quad k \neq j \quad (11.4)$$

with the initial condition

$$S_0^{(1)} = S \quad (11.5)$$

A graphical representation of this set of relationships is shown in Fig. 11.1. Here we build a search-tree. Each node bears the name of a subset  $S_{0lm\dots pk}^{(i)}$  of solutions of  $\mathcal{P}$ . It is convenient to call the superscript  $i$  the level. The set of  $i$  indices  $0lm\dots pk$  indicates a path from the node to which they belong to the root of the tree.

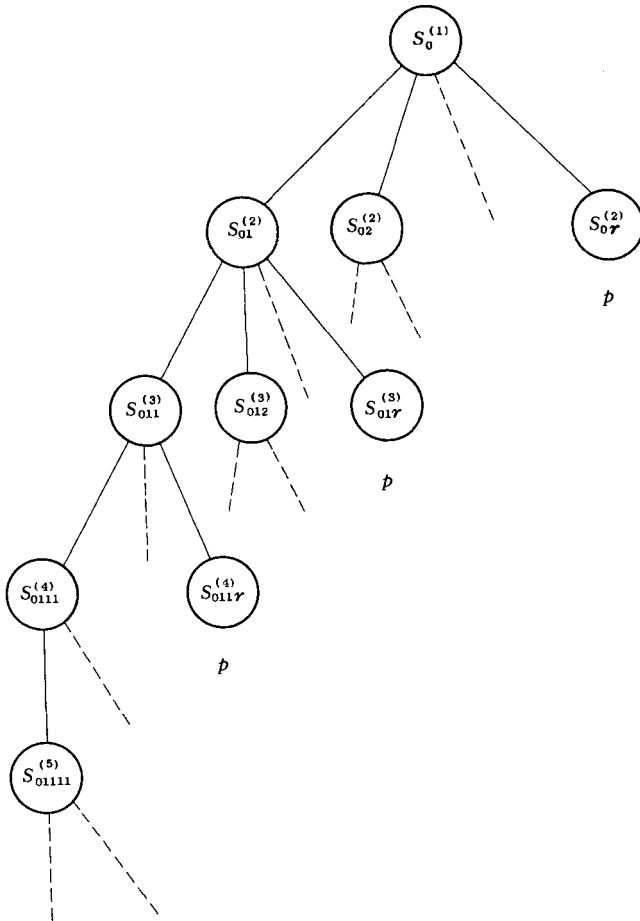


Fig. 11.1. Search tree for problem  $\mathcal{P}$ . (Each partition contains  $r$  subsets;  $p$  indicates pending nodes.)

For example, in Fig. 11.1, node  $S_{011r}^{(4)}$  is at level 4, and the path to the root of the tree is

$$S_{011r}^{(4)}, S_{011}^{(3)}, S_{01}^{(2)}, S_0^{(1)}$$

A *closed* node in the search is a node that can no longer be partitioned (cannot have followers). A *pending* node is one which is not closed. For example, on Fig. 11.1  $S_0^{(1)}$ ,  $S_{01}^{(2)}$  are closed, and  $S_{011r}^{(4)}$  is pending.

Since each subset corresponding to each node is partitioned into two or more nonempty subsets, the cardinality of the subsets is monotonically



decreasing along a branch of the search tree, i.e.,

$$|S_{011r}^{(4)}| < |S_{011}^{(3)}| < \cdots < |S_0^{(1)}| = |S|$$

$|S|$  is finite, so we will eventually reach a level at which one of the pending nodes  $S_{0lm\dots pq}^{(j)}$  contains only one element of  $S$ , i.e., contains a solution to  $\mathcal{P}$ . This is a *terminal* node.

The goal of the method is to get the minimal (or one of the minimal) solution  $\sigma^*$  by enumerating as few nodes as possible. To do so, at each node we must be able to compute, over the subset assigned to the node, an upper and lower bound for the function  $f$ .

The strategy consists of branching from the pending node having the least lower bound. In other words, one uses a property inherent in the nature of the problem to make a partition of the most promising pending node. For a terminal node, the upper and lower bounds collapse to the value of  $f$  for the solution assigned to this node. The search is ended when a node contains a feasible solution, the value of which is less than the smallest value of the lower bounds of the pending vertices.

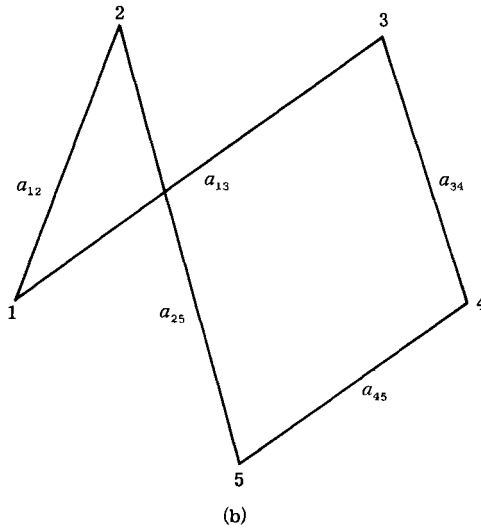
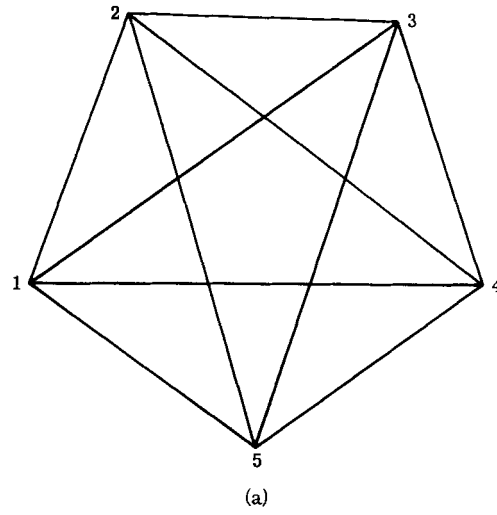
The next section develops an example of the method for the type of problem to which it was first applied.

### The Traveling Salesman Problem

Several examples of branch and bound algorithms will be constructed for the traveling salesman problem to emphasize the fact that branch and bound is not one method, but a class of methods.

The statement of the problem is [1]: "A traveling salesman, starting in one city, wishes to visit each of  $n - 1$  other cities once and only once and return to the start. In what order should he visit the cities to minimize the total distance traveled?"

A *feasible* solution to this problem is called a *tour* (*Hamiltonian cycle*). It is a cycle containing every vertex; a Hamiltonian circuit of the graph corresponding to the geographical map is shown in Fig. 11.2. The function  $f$  to be minimized might be the total distance or the total cost of traveling. For example, a mileage chart is shown in Table 11.1 for a symmetric, traveling salesman problem with  $n = 5$ . In this case the set of feasible solutions contains all the  $4! = 24$  tours, the number of Hamiltonian circuits in the complete graph  $K_5$  (cf. Berge [2, p. 8]).



**Fig. 11.2.** (a) Graph of the connections between the cities— $K_5$ . (b) Example of a Hamiltonian circuit.

TABLE 11.1  
DISTANCES BETWEEN THE CITIES

	2	3	4	5
1	13	3	19	4
2		25	3	5
3			12	10
4				8

METHOD 1

To tour all the cities, the salesman has to make  $n$  trips. Thus we can define any solution  $\sigma_j$  to be a set of  $n$  distinct edges. There are

$$\binom{10}{5} = 252$$

such sets for  $n = 5$ . Let  $S$  be the set of solutions  $\{\sigma_j\}$ . We search for a solution which is a tour and minimizes the sum of the distances corresponding to its edges.

During the search, the *current node* in the node under examination; the *current edge* is an edge temporarily singled out. At any time in the search, an edge of  $Kn$  is said to be *imposed* if it has to be part of all the subsequent solutions, *forbidden* if it cannot enter any subsequent solution, and *free* otherwise. For each node of the search tree we record the imposed edges of the graph  $Kn$  of the problem. If an edge is imposed (forbidden) in a node  $S_{0...kl}^{(i)}$ , it will be imposed or forbidden for all the followers of  $S_{0...kl}^{(i)}$ .

At any time,  $L^0$  is an upper bound corresponding to the particular tour,  $\sigma^0$ .

Algorithm

*Step 1.* Initialization. Pick any tour, call it  $\sigma^0$  and compute its total distance  $L^0$ . This gives an initial upper bound for the problem.

The root of the search tree is assigned the set  $S_0^{(1)} = S$  of all possible solutions. Set the current node to be  $S_0^{(1)}$ . All edges of  $Kn$  are free.

*Step 2.* Computation of a new lower bound. Compute a lower bound  $l$  for the current node by adding up the five smallest entries in the distance

chart corresponding to imposed or free edges of  $Kn$ . Record the corresponding solution  $\sigma$ .

If  $\sigma$  is not a tour, then, go to step 3, otherwise go to 2.1.

*Step 2.1.* If  $l$  is equal to  $L^0$ , or  $l$  is smaller than the smallest of the lower bounds corresponding to the pending nodes, stop; we have an optimal solution.

*Step 2.2.* If  $l$  is smaller than  $L^0$ , set  $L^0$  to  $l$  and  $\sigma^0$  to  $\sigma$ ; go to Step 3.

*Step 3.* Partitioning criterion. Consider the pending node having the smallest lower bound  $l$ . If the corresponding solution  $\sigma$  is a tour, stop; it is the optimal solution; otherwise go on.

*Step 3.1.* If  $\sigma$  contains a vertex of  $Kn$  on degree more than 2, set the current edge  $e$  to be one of the free edges incident to this vertex; go to Step 4.

*Step 3.2.* If  $\sigma$  does not contain a vertex of degree higher than two, but contains subtours, set  $e$  to be one edge of one of the subtours.

*Step 4.* Partitioning. Partition  $S_{0...kl}^{(1)}$  into two nodes, one  $S_{0...kl1}^{(i+1)}$  having  $\sigma$  as a solution and  $e$  as an imposed edge, the other  $S_{0...kl2}^{(i+1)}$  has  $e$  as a forbidden edge.

If in  $S_{0...kl1}^{(i+1)}$  the imposed edges form a subtour or are incident to a vertex of  $Kn$  of degree more than two, close  $S_{0...kl1}^{(i+1)}$ , or equivalently set its lower bound to  $\infty$ , otherwise set the lower bound to  $l$ . Close  $S_{0...kl}^{(i)}$ . Set the current node to be  $S_{0...kl2}^{(i)}$  and go to Step 2.

### *Summary of the Search for the Example with 5 Cities*

(Solution shown in Fig. 11.8)

#### *Iteration 1*

*Step 1.* Take  $\sigma^0$  to be, for example,  $(a_{12}, a_{23}, a_{34}, a_{45}, a_{51})$ . Then  $L^0 = 13 + 25 + 12 + 8 + 4 = 62$ .  $S = S_0^{(1)}$ , all edges of  $K_5$  are free.

*Step 2.* Consider  $S_0^{(1)}$ .

$$\sigma = (a_{13}, a_{24}, a_{25}, a_{15}, a_{45}), \quad l = 23 \text{ (Fig. 11.3)}$$

*Step 3.* Vertex 5 of the graph corresponding to  $\sigma$  (Fig. 11.3) has degree 3; pick  $e = a_{15}$ , for example.

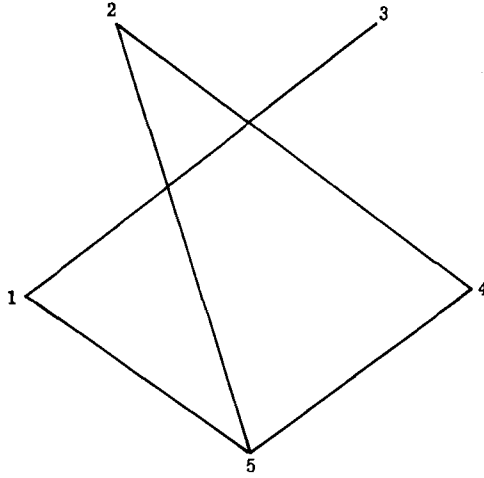


Fig. 11.3. Solution corresponding to the lower bound of  $S_0^{(1)}$ .

*Step 4.* Define

$$S_{01}^{(2)} = \{\sigma_j \in S_0^{(1)} \mid a_{15} \in \sigma_j\}$$

$$S_{02}^{(2)} = \{\sigma_j \in S_0^{(1)} \mid a_{15} \notin \sigma_j\}$$

$S_0^{(1)}$  closed. Current node is  $S_{02}^{(2)}$ . The lower bound is 23.

Iteration 2

*Step 2.* Current node is  $S_{02}^{(2)}$ ;  $a_{15}$  is forbidden.

$$\sigma = (a_{13}, a_{24}, a_{25}, a_{35}, a_{45}), \quad l = 29 \text{ (Fig. 11.4)}$$

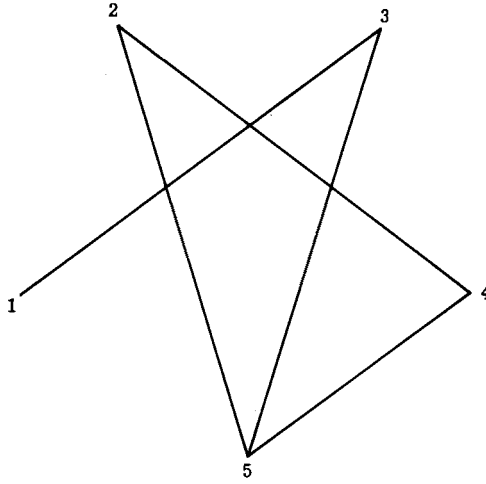
*Step 3.* Lowest lower bound on pending node is 23 for  $S_{01}^{(2)}$ . Vertex 5 of the graph of the solution obtained in step 2 of iteration 1 (Fig. 11.3) has degree 3. Free edges incident to 5 are  $a_{25}$  and  $a_{45}$ ,  $a_{15}$  being imposed. Pick  $e = a_{25}$ , for example.

*Step 4.* Define

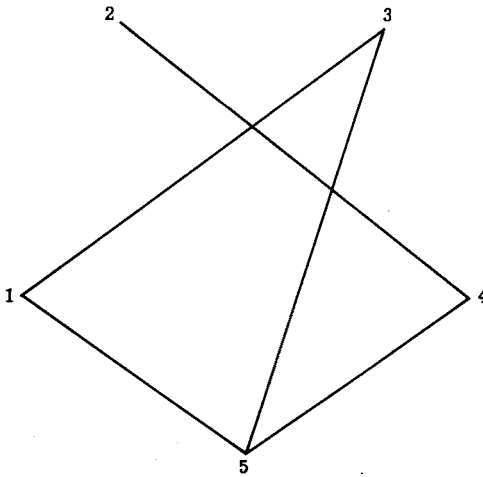
$$S_{011}^{(3)} = \{\sigma_j \in S_{01}^{(2)} \mid a_{25} \in \sigma_j\}$$

$$S_{012}^{(3)} = \{\sigma_j \in S_{01}^{(2)} \mid a_{25} \notin \sigma_j\}$$

$S_{01}^{(2)}$  is closed. Current node is  $S_{012}^{(3)}$ . The lower bound is 23.



**Fig. 11.4.** Solution corresponding to the lower bound of  $S_{01}^{(2)}$ .



**Fig. 11.5.** Solution corresponding to the lower bound of  $S_{012}^{(3)}$ .

## Iteration 3

*Step 2.* Current node is  $S_{012}^{(3)}$ .  $a_{15}$  is imposed,  $a_{25}$  is forbidden.

$$\sigma = (a_{13}, a_{15}, a_{24}, a_{35}, a_{45}), \quad l = 28 \text{ (Figure 11.5)}$$

*Step 3.* Lowest lower bound is 23 on pending node  $S_{011}^{(3)}$ . Vertex 5 of  $K_5$  of the solution obtained in step 2 of iteration 1 (Fig. 11.3) has degree 3, the only free edge incident to 5 is  $a_{45}$ . Set  $e = a_{45}$ .

*Step 4.* Define

$$S_{0111}^{(4)} = \{\sigma_j \in S_{011}^{(3)} \mid a_{45} \in \sigma_j\}$$

$$S_{0112}^{(4)} = \{\sigma_j \in S_{011}^{(3)} \mid a_{45} \notin \sigma_j\}$$

A lower bound for  $S_{0111}^{(4)}$  is infinity since  $a_{15}a_{25}a_{45}$  are imposed and cannot lead to a feasible solution.  $S_{0111}^{(4)}$  is closed.  $S_{011}^{(3)}$  is closed. Current node is  $S_{0112}^{(4)}$ .

## Iteration 4

*Step 2.* Current node is  $S_{0112}^{(4)}$ ;  $a_{15}, a_{25}$  are imposed,  $a_{45}$  is forbidden.

$$\sigma = (a_{13}, a_{15}, a_{24}, a_{25}, a_{35}), \quad l = 25 \text{ (Fig. 11.6)}$$

*Step 3.* Lowest lower bound is 25 corresponding to  $S_{0112}^{(4)}$ . Since vertex 5 is of degree 3 (Fig. 11.6) and the only free edge incident to 5 is  $a_{35}$ , set  $e = a_{35}$ .

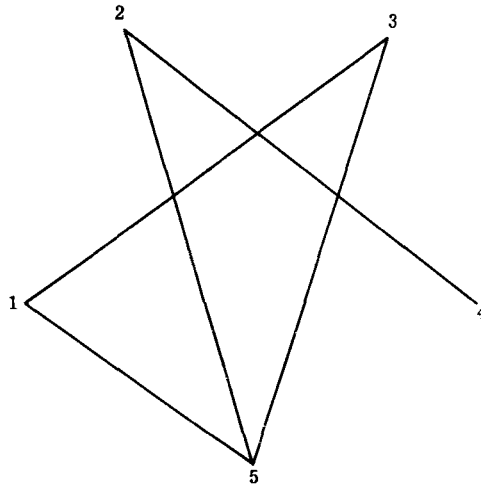


Fig. 11.6. Solution corresponding to the lower bound of  $S_{0112}^{(4)}$ .

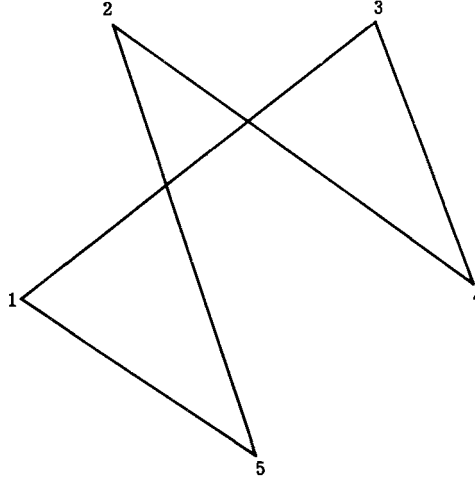


Fig. 11.7. Solution corresponding to the lower bound of  $S_{01122}^{(5)}$ .

Step 4. Define

$$S_{01121}^{(5)} = \{\sigma_j \in S_{0112}^{(4)} \mid a_{35} \in \sigma_j\}$$

$$S_{01122}^{(5)} = \{\sigma_j \in S_{0112}^{(4)} \mid a_{35} \notin \sigma_j\}$$

A lower bound for  $S_{01121}^{(5)}$  is infinity, since  $a_{15}$ ,  $a_{25}$ ,  $a_{35}$  being imposed, cannot lead to a feasible solution.

Iteration 5

Step 2. Current node is  $S_{01122}^{(5)}$ ;  $a_{15}$ ,  $a_{25}$  are imposed  $a_{35}$ ,  $a_{45}$  are forbidden.

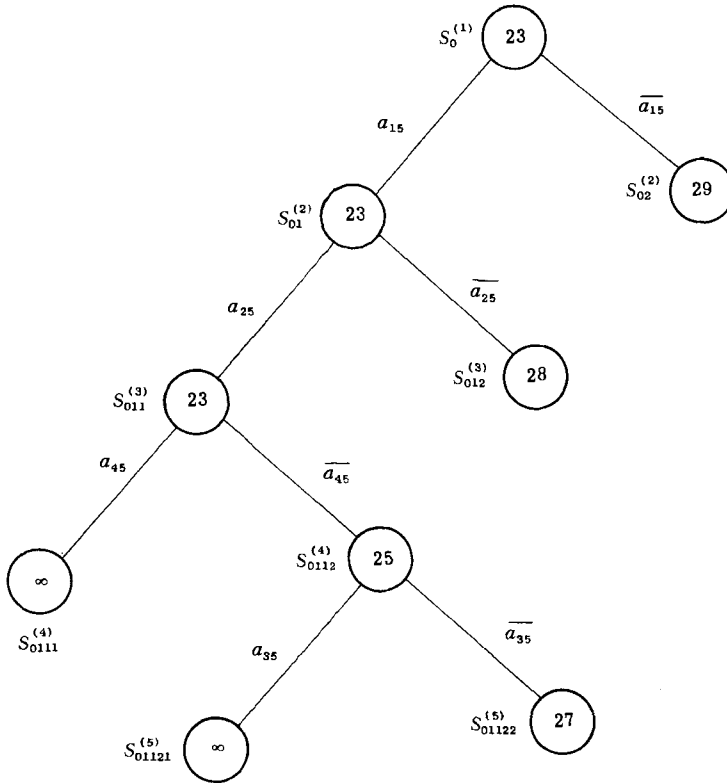
$$\sigma = (a_{13}, a_{15}, a_{24}, a_{25}, a_{34}), \quad e = 27 \text{ (Fig. 11.7)}$$

$\sigma$  is a tour; 27 is lower than the lower bounds 28 and 29, therefore, we have an optimal solution (Fig. 11.8).

## METHOD 2

Another approach is to restrict the solution  $\sigma_j$  to *partial tours*, i.e. sets of edges of  $Kn$  which do not contain any vertex of degree more than two and any circuit of length less than  $n$ . Then one branches from each pending node by adding to the partial tour already obtained the shortest (or least





**Fig. 11.8.** Search tree. (The encircled numbers are the lower bounds. Newly imposed ( $a_{ij}$ ) or forbidden ( $\overline{a_{ij}}$ ) edges are indicated.)

expensive) compatible edge, i.e., an edge such that the resulting new set is a partial tour. To carry out this procedure we list the edges by increasing cost, as in Table 11.2. The root of the search tree is then partitioned into  $n(n-1)/2$  subsets corresponding to each partial tour built by taking as the first edge one of the edges in the list, and then adding edges further down in the list. Thus for our example (Fig. 11.9)  $S_{01}^{(2)}$  would contain solutions having  $a_{13}$  imposed, and edges below  $a_{13}$  in the list (Table 11.2).  $S_{02}^{(2)}$  would contain solutions having  $a_{24}$  imposed and edges below  $a_{24}$  in the list, i.e. solutions of  $S_{02}^{(2)}$  will not contain edge  $a_{13}$ , etc.

A lower bound can be computed for the solution class of node  $S_{0p\dots q}^{(k+1)}$  of the search tree by adding the costs of the first  $k$  imposed edges to the costs of the  $n-k$  edges following the last edge chosen from the list. For example, a lower bound for  $S_{01}^{(2)}$  is  $(3) + (3 + 4 + 5 + 8) = 23$ ; a lower

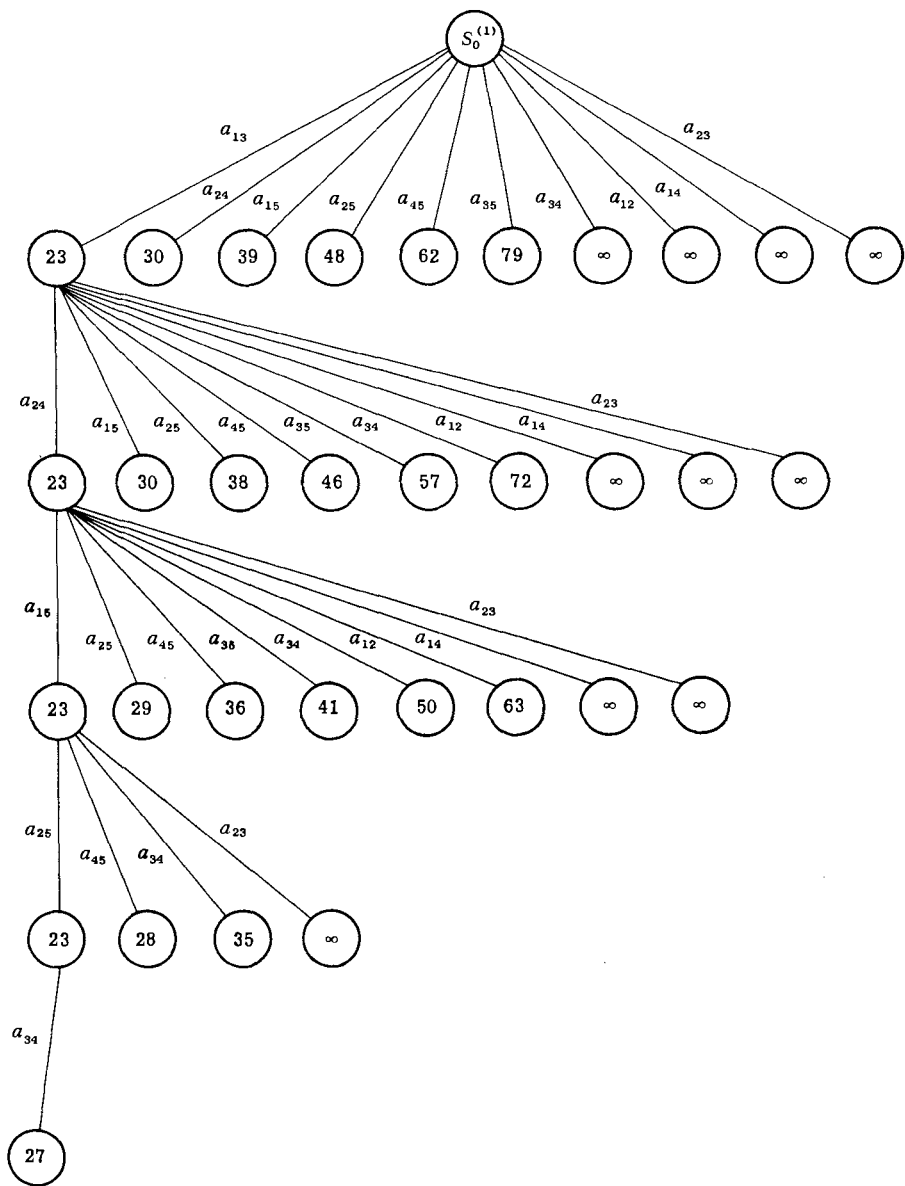


Fig. 11.9. Search tree, method 2.

TABLE 11.2  
RANKING OF THE EDGES

Rank	Edge	Cost
1	$a_{13}$	3
2	$a_{24}$	3
3	$a_{15}$	4
4	$a_{25}$	5
5	$a_{45}$	8
6	$a_{35}$	10
7	$a_{34}$	12
8	$a_{12}$	13
9	$a_{14}$	19
10	$a_{23}$	25

bound for  $S_{02}^{(2)}$  is  $(3) + (4 + 5 + 8 + 10) = 30$ , etc. If, at the bottom of the list, not enough edges remain to compute a lower bound, then the bound is set to infinity.

During the search we always branch from the smallest lower bound. At the beginning of the search an upper bound may be infinity, or else the cost of any tour. The upper bound is updated as in method 1, when tours are found. The terminal criteria are the same as in method 1.

The complete search tree is shown on Fig. 11.9 for the example corresponding to the mileage chart of Table 11.1 (which gave rise to the list of Table 11.2).

Notice that the number of nodes in the search tree can be prohibitively high if  $n$  is large. This can be reduced by partitioning each pending node into only two new nodes  $S_{0k\dots m1}^{(i)}$  and  $S_{0k\dots m2}^{(i)}$ , with

$$S_{0k\dots m1}^{(i)} = S_{0k\dots m1}^{(i)}$$

$$S_{0k\dots m2}^{(i)} = \bigcup_{p=2}^q S_{0k\dots mp}^{(i)}$$

We assign to  $S_{0k\dots m2}^{(i)}$  the lower bound corresponding to  $S_{0k\dots m2}^{(i)}$ . The new search tree (Fig. 11.10) is more compact and requires the computation of only two bounds at each level.

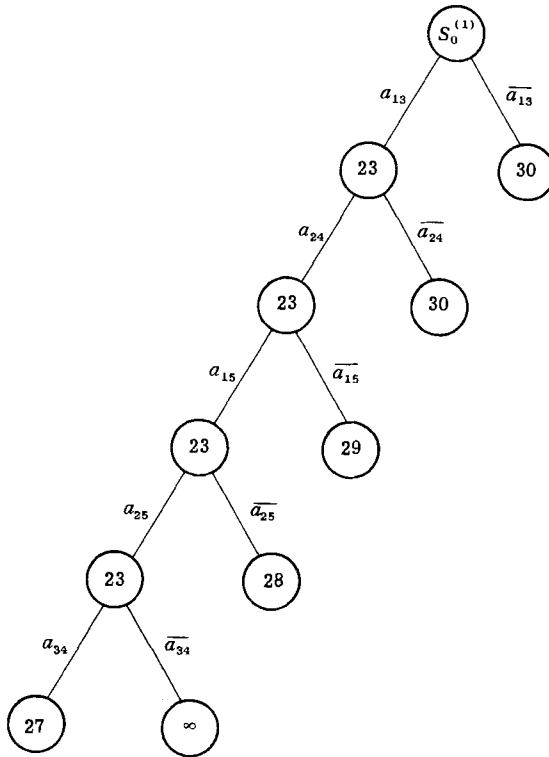


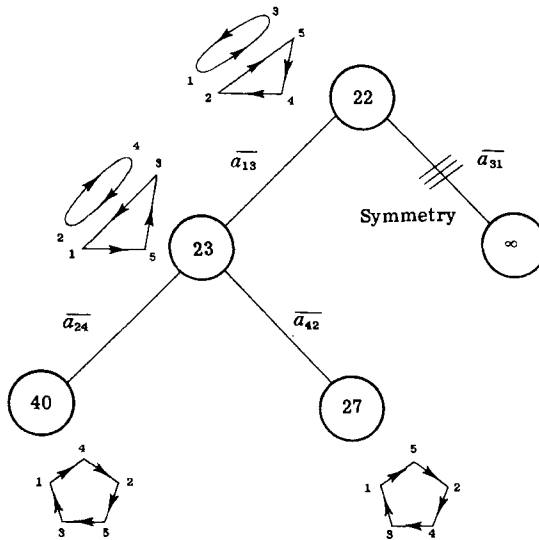
Fig. 11.10. Compact search tree for method 2.

Notice that the search tree in Fig. 11.10 is very similar to the tree of method one. The difference lies in that each solution in method two has to be a partial tour.

### METHOD 3

Method 3 which is a sophisticated version of method 1, was proposed by Eastman [3]. Unlike in method 1, where a solution  $\sigma_j$  may be any set of  $n$  edges of  $Kn$ ,  $\sigma_j$  has to be a solution to the assignment problem (cf. Hillier and Lieberman [4, p. 198]), corresponding to the matrix of costs (or distances).  $\sigma_j$  does not contain any vertex of degree higher than 2, but may contain subtours, each edge of which will become a forbidden edge when branching. There are no imposed edges. The cost attached to the

solution of the assignment problem yields a lower bound for each node. The corresponding search tree is shown in Fig. 11.11. Since our example problem is symmetric, and the method was derived for unsymmetric cost matrices, one can remove the second half of the search tree ( $S_{02}^{(2)}$  and followers) from further considerations. Although there are only four nodes

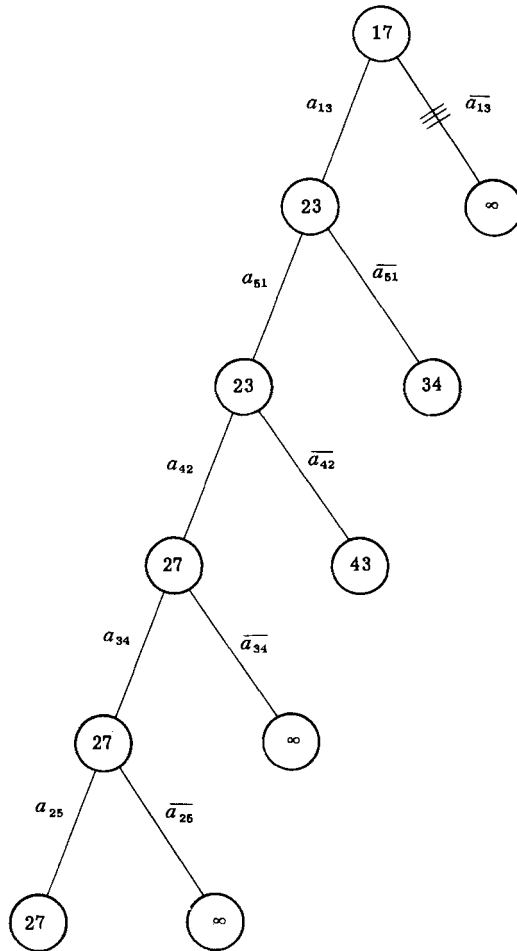


**Fig. 11.11.** Search tree for method 3. (Solutions to the corresponding assignment problems are shown next to each node.  $\overline{a_{ij}}$  are forbidden edges. Encircled numbers are lower bounds.)

in the search tree, this does not mean that method 3 is best, since at each branching step, one has to solve the assignment problem. Method 1 and 3 go under the name of *subtour elimination procedures*.

#### METHOD 4

Method 4, the approach of Little *et al.* [1], is essentially the same as method 2. The computation of bounds is more complicated, but more efficient, and some heuristic rules determine the choice of the imposed edges. The search tree is shown in Fig. 11.12. Method 2 and 4 belong to the class of "tour building" algorithms.



**Fig. 11.12.** Search tree for method 4. (As in method 3, one can close  $S_{02}^{(2)}$  for symmetric problems.)

### Integer Programming and Pseudo-Boolean Programming

Many combinatorial problems that arise in engineering or management may be formulated either as integer programming problems; or (in the context of decision making) as pseudo-Boolean problems.

An integer programming problem may be stated as:

$$\text{Maximize } \sum_{j=1}^n c_j x_j \quad (11.6)$$

Subject to

$$\sum_{j=1}^n a_{ij}x_j \leq b_i, \quad i = 1, 2, \dots, m \quad (11.7)$$

$$x_j \in N^* \quad (\text{set of nonnegative integers}) \quad (11.8)$$

A pseudo-Boolean problem differs in statement (11.8), which should be replaced by

$$x_j \in \{0, 1\} \quad (11.9)$$

Many algorithms have been proposed to solve these problems, but no particular one provides complete satisfaction in all difficult or degenerate cases. This is primarily due to the restriction of the variables to integer values. To overcome this, Land and Doig [5] proposed to use branch and bound by relaxing the constraint (11.8) and allowing solutions  $\sigma_j$  to contain non-integral values. A *feasible* solution to  $\mathcal{P}$  is an integral solution (obeying (11.8)). To obtain a lower bound for the node  $S_0^{(1)}$  we solve the linear programming problem corresponding to (11.6) and (11.7). If the solution is feasible, then the optimal  $\sigma^*$  has been found, otherwise one of the variables with noninteger values is used to partition the set of solutions. For example, if  $x_6$  has a value of 10.2,  $S_{01}^{(2)}$  will represent solutions to the problem defined by (11.6), (11.7) and

$$x_6 \leq 10 \quad (11.10)$$

$S_{02}^{(2)}$  represents the solutions to the problem defined by (11.6), (11.7) and

$$x_6 \geq 11 \quad (11.11)$$

It is easy to see that this property partitions the solutions of  $\mathcal{P}$ .

The process is repeated at each level, with one of the non-integer variables of the solution corresponding to the upper bound of the previous node. The algorithm terminates when an all-integer solution is obtained with a value less than the upper bounds of the pending nodes.

A different approach can be used when the variables are bounded. We then replace each variable by its binary expansion and convert the problem into a pseudo-Boolean programming problem.

For example, if  $6 \leq x_5 \leq 12$  one may write  $x_5$  as  $x_5 = 6 + y_{50} + 2y_{51} + 4y_{52}$ .

The corresponding methods for pseudo-Boolean problems are known as implicit enumeration algorithms (or sometimes additive algorithms). The idea is to progressively set variables to 0 or 1, building *partial solutions*

and computing appropriate bounds until one obtains a feasible solution of minimum cost. In the process we use the constraints to discard possible *completions* (complement of a partial solution) which are not feasible. This combines the features of branch and bound and branch and prune methods. The search tree contains  $(2^{n+1} - 1)$  nodes and the last level represents the  $2^n$  possible solutions to the problem. Every time one *prunes* a branch (*closes* a node) at level  $k$  (corresponding to a partial solution of cardinality  $k$ ), one effectively condemns  $2^{n-k}$  terminal nodes. The choice of which variable to set first is highly debatable, and no one has yet found a satisfactory answer. Usually a set of heuristical rules to guide this choice must be formulated.

As an illustration, we solve the problem given as example 1 of a paper by Balas [6].

Minimize

$$5x_1 + 7x_2 + 10x_3 + 3x_4 + x_5 \quad (11.12)$$

Subject to

$$-x_1 + 3x_2 - 5x_3 - x_4 + 4x_5 \leq -2 \quad (11.13)$$

$$2x_1 - 6x_2 + 3x_3 + 2x_4 - 2x_5 \leq 0 \quad (11.14)$$

$$x_2 - 2x_3 + x_4 + x_5 \leq -1 \quad (11.15)$$

An upper bound for the problem is 26; a lower bound 0. To minimize the objective function, it seems reasonable to try to obtain solutions for which the values of variables associated with large cost coefficients are zero. We, therefore try to set  $x_3$  to 0. This corresponds to the partition  $(S_{01}^{(2)}, S_{02}^{(2)})$  of  $S_0^{(1)}$  in the search tree. But now there is no completion which will render the constraint (11.15) feasible; therefore we have to discard this class of solutions and close  $S_{01}^{(2)}$ . The lower bound to the problem is now 10. The solution class associated with node  $S_{02}^{(2)}$  satisfies

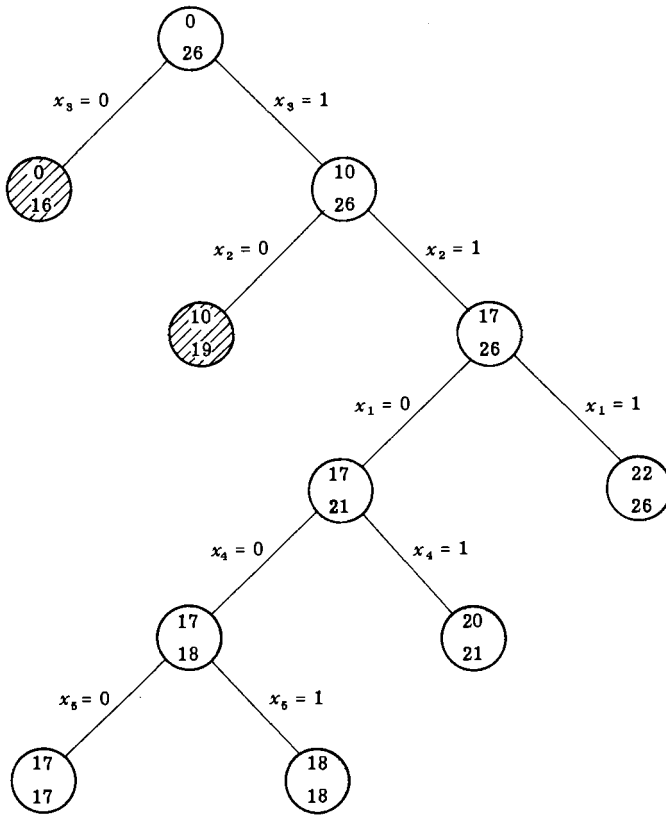
$$-x_1 + 3x_2 - x_4 + 4x_5 \leq 3 \quad (11.16)$$

$$2x_1 - 6x_2 + 2x_4 - 2x_5 \leq -3 \quad (11.17)$$

$$x_2 + x_4 + x_5 \leq 1 \quad (11.18)$$

Using the same heuristic reasoning, we can try to set  $x_2$  to zero, since  $x_2$  now has the largest cost coefficient. But, again there is no completion which makes constraint (11.17) feasible. We close  $S_{02}^{(3)}$  and change the





**Fig. 11.13.** Example of search tree for pseudo-Boolean programming. (Shaded circles indicate closed nodes (no feasible completion); l: lower bound; u: upper bound.)

lower bound to a value of 17. We repeat the process with variables  $x_1$ ,  $x_4$  and  $x_5$  and find a feasible solution having a value of 17 equal to the lower bound, therefore we stop. The corresponding search tree has been drawn; Fig. 11.13.

Several codes have been written for integer programming as well as pseudo-Boolean programming and the results appear in the literature [7–9]. The program described by Benayoun *et al.* [9] for mixed integer programming can handle several thousand constraints and a few hundred discrete variables. It was written for a CDC 6600.

To be complete we also cite a recent Boolean approach to this type of problem [10].

### Generalization of the Branch and Bound Method

Branch and bound is a structural method containing the seeds of many possible generalizations. One of them is to base the choice of a node from which to branch not on the value of its lower bound, but rather on a special function  $g$  which provides a local evaluation of the situation.  $g$  has a wide use in artificial intelligence in gameplaying and theorem-proving programs. It is referred to as the *scoring* or *evaluation* function. An illustration is now provided in the form of the “bidding contest” problem formulated by Dr. Werner Burckhardt.

### Bidding Contest Problem

A group desiring to build a facility subdivides the facility into several “lots.” The facility may be a building if the group is a university, engineering equipment if the group is a chemical company, etc. A number of contractors bid on the lots. They may bid on single lots and/or combinations thereof. Table 11.3 shows a problem with eight bidders and four lots; in the second row, contractor 3 bids exclusively on the combination of lots 1 and 2, contractor 4 bids on any combination of lots 2, 3, and 4. A particular combination will be called an *offer*. All the offers are summarized in Table 11.4 and Table 11.5 gives the quantity discounts the contractors are ready to concede in relation to the cost of the accepted offer. For example, contractor C is ready to give a 4% discount for prices ranging from 200 included to 400 excluded. The problem for the group is to find the ten best feasible, cost-minimal, contractors-lots combinations.

Minimizing the final cost is equivalent to maximizing the total discount, so we can list the offers by decreasing average discount (Table 11.6). The total discount for an offer is the difference between its cost and the corresponding sum of maximum prices of the lots in the offer. The function  $f$  to minimize is the total discount, the evaluation function  $g$  is the average discount per lot.

The idea is to build a search tree having as few nodes as possible, and to make as few computations as possible by obtaining a suitable branching criterion. The classical branch and bound technique would lead us to perform a partition at each stage by considering the solutions that contain a certain offer  $B$ , and those which do not, and to compute an upper bound for each class of solutions (maximization problem). The departure from this approach is that we do not compute upper bounds, but rather use the

TABLE 11.3  
BIDDING STRUCTURE<sup>a</sup>

<div>Contractor</div> <div></div>	Lot			
	1	2	3	4
<i>A</i>	X			
<i>B</i>	0	0		
<i>C</i>	X	X		
<i>D</i>	X	X	X	X
<i>E</i>		X	X	X
<i>F</i>		0	0	
<i>G</i>	0	0	0	0
<i>H</i>		X	X	X

<sup>a</sup> Rows containing 0 lead to only one bid, rows containing X lead to all possible combinations of lots.

TABLE 11.4  
ACTUAL PRICES (WITHOUT DISCOUNT) IN 10<sup>3</sup> DOLLARS

<div>Contractor</div> <div></div>	Lot			
	1	2	3	4
<i>A</i>	190			
<i>B</i>	190	400		
<i>C</i>	200	400		
<i>D</i>	220	440	230	320
<i>E</i>		440	220	330
<i>F</i>		400	210	
<i>G</i>	200	400	200	300
<i>H</i>		420	210	320

TABLE 11.5  
DISCOUNT RATES (%)

Contractor \ Cost (10 <sup>3</sup> \$)	Discount Rates (%)				
	200-400-	400-600-	600-800-	800-1000-	1000-∞
<i>A</i>	0	0	0	0	0
<i>B</i>	2	2	2	2	2
<i>C</i>	2	4	6	6	6
<i>D</i>	2	4	6	8	10
<i>E</i>	2	4	7	9	9
<i>F</i>	0	0	0	0	0
<i>G</i>	1	1	1	1	1
<i>H</i>	2	3	4	5	5

TABLE 11.6  
MAIN LIST OF OFFERS

Rank	Offer	Discount ( <i>f</i> )	Average discount ( <i>g</i> )	Rank	Offer	Discount ( <i>f</i> )	Average discount ( <i>g</i> )
1	<i>C2</i>	56	56	19	<i>H3</i>	24.2	24.2
2	<i>C12</i>	96	48	20	<i>C1</i>	24	24
3	<i>B12</i>	81.8	40.9	21	<i>H34</i>	45.9	22.95
4	<i>E234</i>	99.1	33.0-	22	<i>D23</i>	40.2	20.1
5	<i>D1234</i>	131	32.75	23	<i>D12</i>	39.6	19.8
6	<i>G1234</i>	131	32.75	24	<i>D134</i>	56.2	18.7-
7	<i>H23</i>	65.2	32.6	25	<i>E2</i>	17.6	17.6
8	<i>H2</i>	32.6	32.6	26	<i>D4</i>	16.4	16.4
9	<i>H234</i>	97.5	32.5	27	<i>H4</i>	16.4	16.4
10	<i>F23</i>	60	30	28	<i>E34</i>	32	16
11	<i>A1</i>	30	30	29	<i>D34</i>	32	16
12	<i>H24</i>	59.6	29.8	30	<i>D14</i>	31.6	15.8
13	<i>D234</i>	89.2	29.7-	31	<i>E3</i>	14.4	14.4
14	<i>D124</i>	88.4	29.5+	32	<i>D13</i>	18	9
15	<i>E23</i>	56.2	28.1	33	<i>E4</i>	6.6	6.6
16	<i>D24</i>	55.6	27.8	34	<i>D2</i>	5.6	5.6
17	<i>D123</i>	71.2	27.1-	35	<i>D3</i>	4.6	4.6
18	<i>E24</i>	53.9	26.95	36	<i>D1</i>	4.4	4.4

new average discount attached to the class of solutions that contains  $B$ , and assign it as *indicator* to each of the two newly-created nodes. This value does *not* constitute an upper bound. Also, the cumulated offers, the number of lots they contain, and the corresponding discount subtotals are recorded for each node. (This information need not be stored, but may be reconstructed as needed, by backtracking to the root of the tree.)

We then branch from the node having the largest indicator and store the solutions until we have 10 feasible ones, at which point we use a *pruning* criterion. When a node  $S_{0k...mn}^{(i)}$  is selected for branching the choice of the offer to be inserted is determined by the list of Table 11.6. If  $B$  is the offer which leads to the addition of  $S_{0k...mn}^{(i)}$  to the search tree, the next offer  $B'$  to be added to partition  $S_{0k...mn}^{(i)}$  is the next *compatible* offer following  $B$  in the list. Compatible meaning here that the intersection of the contractor and set of lots of  $B'$ , and the sets of contractors and lots corresponding to the solution class of  $S_{0k...mn}^{(i)}$ , is empty. Or, equivalently, that the contractor  $B'$  has not already been chosen and the lots are not already selected in the previous bids in this branch of the search tree. If we come to the end of the list, the node  $S_{0k...mn}^{(i)}$  is closed and/or its evaluation function set to infinity. The search tree corresponding to this phase of the problem is shown in Fig. 11.14. A more compact version of the same tree is given as Fig. 11.15. The details of the first 23 steps are recorded in Table 11.7.

As soon as we have 10 feasible solutions, we rank them in a pushdown list and the value of the tenth offer  $\sigma^*$  yields a lower bound for pruning purposes. This lower bound may be translated in terms of an evaluation function value  $g^*$ .

For example consider a pending node  $S_{0k...mn}^{(i)}$  corresponding to a value of the evaluation function of  $g^0 = c/k$ ,  $c$  being the subtotal for the  $k$  bids recorded in the path from the root to  $S_{0k...mn}^{(i)}$ . To obtain a feasible solution from  $S_{0k...mn}^{(i)}$ , we add offers having evaluation functions of the form  $d/p$ ,  $p$  being the number of lots and  $d$  the total discount for these lots. In order to get a better solution than  $\sigma^*$  in this branch, it is necessary to have

$$\begin{aligned} g^* &\leq \frac{c + d + \dots + e}{k + p + \dots + r} = \frac{1}{N} \left( c + p \left( \frac{d}{p} \right) \right) + \dots + r \left( \frac{e}{r} \right) \\ &\leq \frac{1}{N} \left( c + (N - k) \frac{d}{p} \right) \end{aligned} \quad (11.19)$$

since  $p + \dots + r = N - k$  and  $(d/p) \geq \dots \geq (e/r)$ ,  $N$  being the total number of bids, this can be rewritten as

$$\frac{d}{p} \geq g^* - \frac{k}{N - k} \left( \frac{c}{k} - g^* \right) \quad (11.20)$$

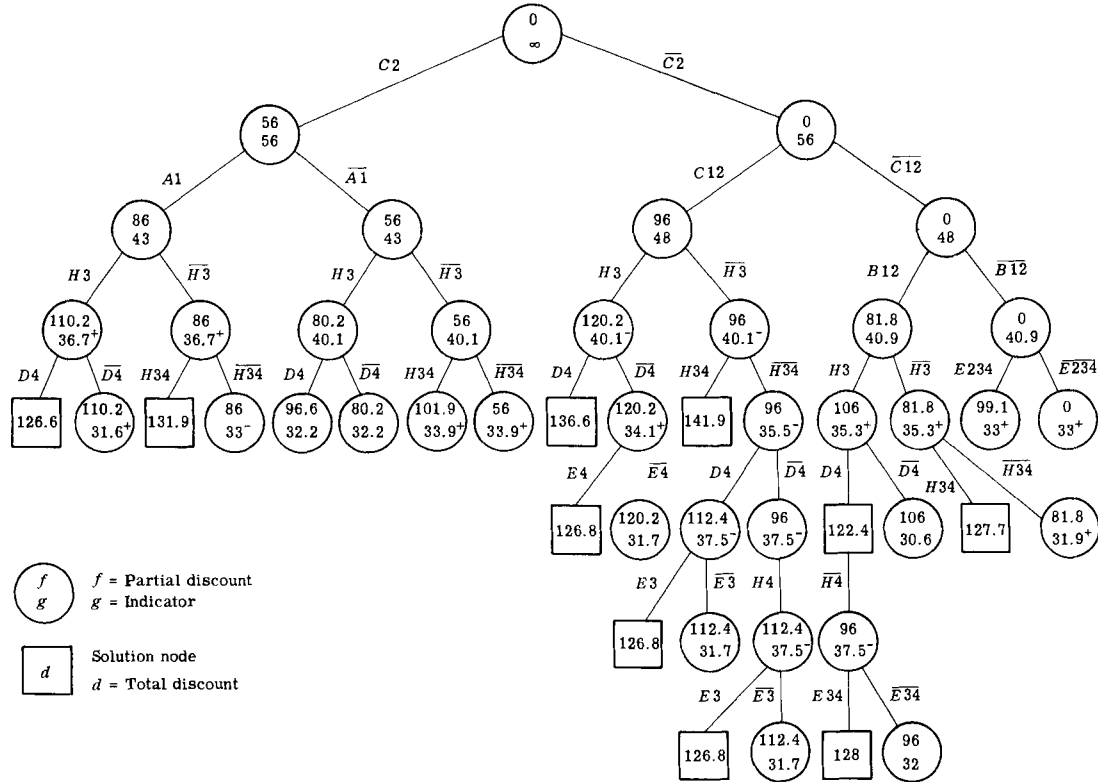


Fig. 11.14. Bidding search tree. (By convention, two nodes of the same partition have the same indicator;  $B_{i,\dots,k}$ , offer corresponding to contractor  $B$  bidding on lots  $i, \dots, k$ , offer is imposed;  $\bar{B}_{i,\dots,k}$ , same offer, forbidden.)

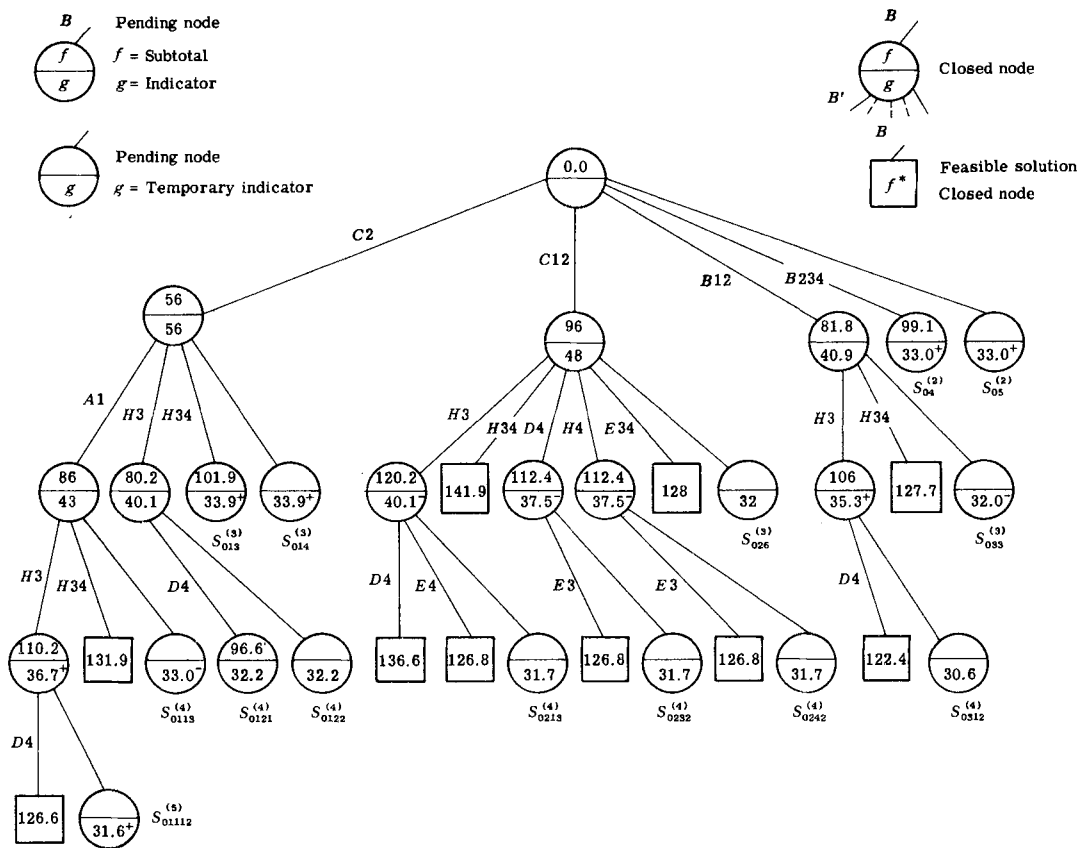


Fig. 11.15. "Compact" search tree for the first phase of the algorithm.

TABLE 11.7  
FIRST PHASE OF THE ALGORITHM

Steps	Formed nodes	Solution	Cumulated bids	Total discount	Average discount
1	$S_{01}^{(3)}$		C2	56	56
2	$S_{011}^{(3)}$		C2A1	86	43
3	$S_{02}^{(2)}$		C12	96	48
4	$S_{021}^{(3)}$		C12H3	120.2	40.1 <sup>-</sup>
5	$S_{022}^{(3)}$		B12	81.8	40.9
6	$S_{0111}^{(4)}$		C2A1H3	110.2	36.7 <sup>+</sup>
7	$S_{0121}^{(4)}$		C2H3	80.2	40.1
8	$S_{02211}^{(5)}$		B12H3	106.0	35.3 <sup>+</sup>
9	$S_{02221}^{(5)}$		E234	99.1	33.0 <sup>+</sup>
10	$S_{01211}^{(5)}$		C2H3D4	96.6	32.2
11	$S_{01221}^{(5)}$		C2H34	101.9	33.9 <sup>+</sup>
12	$S_{02111}^{(5)}$	Yes	C12H3D4	136.6	34.1 <sup>+</sup>
13	$S_{02121}^{(5)}$	Yes	C12H34	141.9	35.5 <sup>-</sup>
14	$S_{01111}^{(5)}$	Yes	C2A1H3D4	126.6	31.6 <sup>+</sup>
15	$S_{01121}^{(5)}$	Yes	C2A1H34	131.9	33.0 <sup>-</sup>
16	$S_{021221}^{(5)}$		C12D4	112.4	37.5 <sup>-</sup>
17	$S_{0212211}^{(7)}$	Yes	C12D4E3	126.8	31.7
18	$S_{0212221}^{(7)}$		C12H4	112.4	37.5 <sup>-</sup>
19	$S_{02122211}^{(8)}$	Yes	C12D4E3	126.8	31.7
20	$S_{02122221}^{(8)}$	Yes	C12E34	128	32
21	$S_{022111}^{(6)}$	Yes	B12H3D4	122.4	30.6
22	$S_{022121}^{(6)}$	Yes	B12H34	127.7	32.0
23	$S_{021121}^{(7)}$	Yes	C12H3E4	126.8	31.7



or

$$g^1 \geq l \quad (11.21)$$

with

$$g^1 = d/p \quad (11.22)$$

and

$$l = g^* - \lambda(g^0 - g^*) \quad (11.23)$$

where

$$\lambda = k/(N - k) \quad (11.24)$$

Since we ordered our offers by decreasing average discount, the value of the evaluation function is monotonically decreasing along each branch of the search tree. We can use (11.21) as a pruning criterion, because  $l$  is a lower bound for  $g$  with respect to the offers to be added at a pending node.

The strategy is to consider the pending node  $S_{0k...mn}^{(i)}$  having the highest value  $g^0$  of the evaluation function  $g$ . If  $g^0$  is smaller than  $g^*$  (corresponding to the tenth offer), the problem terminates. If  $g^0$  is a temporary indicator (as defined on Fig. 11.15), replace it with the value of  $g$  for the predecessor  $S_{0k...m}^{(i-1)}$  of  $S_{0k...mn}^{(i)}$ . Compute  $l$  by (11.23), then try to partition  $S_{0k...mn}^{(i)}$  by adding a compatible offer  $B'$  from Table 11.6. If the value  $g^1$  of the evaluation function for  $B'$  is smaller than  $l$ , then close the node  $S_{0k...mn}^{(i)}$ ; otherwise partition the node as usual.

If a new solution is found, update the list of the 10 temporary best solutions and change  $g^*$  accordingly. Iterate until all nodes have been closed or the problem has terminated.

This final phase of the procedure is summarized on Table 11.8 corresponding to the final "compact" tree shown on Figure 11.16. This problem has 126 solutions, which would require some time to generate in a non-redundant fashion. The advantage of the branch and bound technique is that we generate only 15 feasible solutions which have very good total discount values, by examination only a few nodes of the search tree. The ten best solutions are given in Table 11.9.

Another possible generalization for this class of problem is to use search structures more general than trees. In particular such new structures would include the concept of AND/OR trees, used by Nilsson [11] to solve problems in artificial intelligence. The root of the tree represents a problem with a goal (prove a theorem, draw a flow sheet, win a chess game, etc.). Each node is partitioned into a new set of nodes representing subproblems with subgoals. Certain subsets of these nodes may correspond to problems such

TABLE 11.8  
FINAL RESULTS, BIDDING PROBLEM

Steps	Nodes of the tree of Figs. 11.14 and 11.15	$g^*$	$g^0$	$l$	$g^1$	Cumulated bids	Discount $f$	Solution	New $g^*$
24	$S_{013}^{(3)}$	30.6	33.9	27.3	4.4	Close node			
25	$S_{014}^{(3)}$	30.6	56	22.1 <sup>+</sup>	18.7	Close node			
26	$S_{04}^{(2)}$	30.6	33.0	23.4	30	$E234A1$	129.1	11th	31.6
27	$S_{05}^{(2)}$	31.6	0	31.6	32.75	$D1234$	131	12th	31.7
28	$S_{0113}^{(4)}$	31.7	43	20.4	17.6	Close node			
29	$S_{06}^{(2)}$	31.7	0	31.7	32.75	$G1234$	131	13th	31.7
30	$S_{07}^{(2)}$	31.7	0	31.7	32.6	$H23$	65.2		
31	$S_{07}^{(2)}$	31.7	32.6	30.8	30	Close node			
32	$S_{08}^{(2)}$	31.7	0	31.7	32.6	$H2$	32.6		
33	$S_{08}^{(2)}$	31.7	32.6	31.4	30	Close node			
34	$S_{09}^{(2)}$	31.7	0	31.7	32.5	$H234$	97.5		
35	$S_{09}^{(2)}$	31.7	32.5	29.3	30	$H234A1$	127.5	14th	31.7
36	$S_{010}^{(2)}$	31.7	0	31.7	30	Close node			
37	$S_{0121}^{(4)}$	31.7	32.2	30.2	—	Close node			
38	$S_{0122}^{(4)}$	31.7	40.1	23.3	15.8	Close node			
39	$S_{026}^{(3)}$	31.7	48	15.4	16	$C12D34$	128	15th	31.9
40	$S_{027}^{(3)}$	31.9	48	15.8	14.4	Close node			
41	$S_{033}^{(3)}$	31.9	40.9	22.9	16.4	Close node			
42	$S_{092}^{(3)}$	Stop							

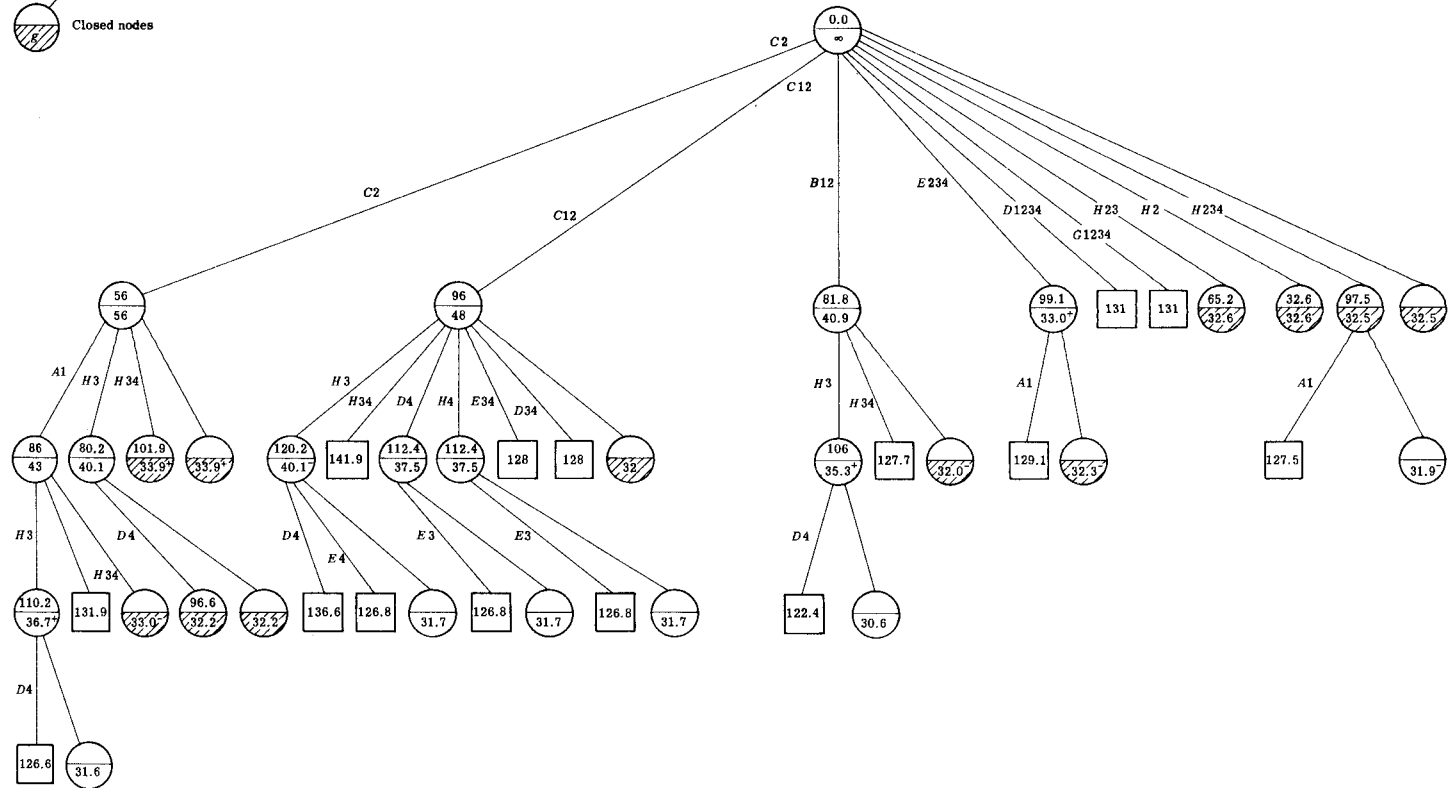


Fig. 11.16. Search tree for the bidding problem.

TABLE 11.9  
SOLUTION TO THE BIDDING CONTEST PROBLEM

Rank	Contractors	Lots	Price	Discount	Final price
1	C	1 2	600	36	1078.1
		H 3 4	530	15.9	
2	C	1 2	600	36	1083.4
	D	4	320	6.4	
	H	3	210	4.2	
3	A	1	190	0	1088.1
		2	400	16	
		H 3 4	530	15.9	
4	D	1 2 3 4	1210	121	1089
5	G	1 2 3 4	1100	11	1089
6	A	1	190	0	1090.9
		E 2 3 4	990	89.1	
7	C	1 2	600	36	1092
		E 3 4	550	22	
8	C	1 2	600	36	1092
		D 3 4	550	22	
9	B	1 2	590	11.8	1092.3
		H 3 4	530	915.9	
10	A	1	190	0	1092.5
		H 2 3 4	950	47.5	

that all of them have to be solved to obtain a feasible solution. This is indicated in Fig. 11.17 by an angle in the branches of  $S_{03}^{(2)}S_{04}^{(2)}S_{05}^{(2)}$  or  $S_{012}^{(3)}S_{013}^{(3)}$ .

One can also envision more general graphs, possibly containing circuits or progressively finite instead of being finite (cf. Berge [2, p. 17]). Possible applications lie in the domain of computerized decision making, computer aided design [12], etc.

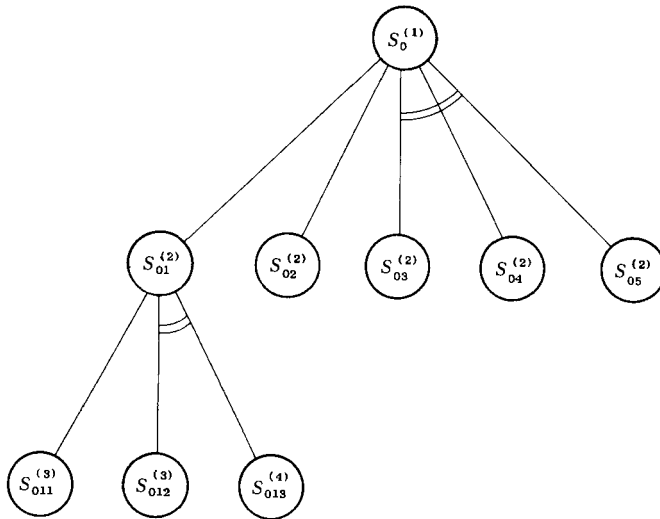


Fig. 11.17. AND/OR tree.

## Conclusion

The traveling salesman example illustrated the compromises between the length of the computations at each node (suboptimization) and the number of nodes of the search tree that have to be generated in order to obtain an optimal solution. It is also important to thoroughly understand the structure of a particular problem before designing a branch and bound scheme to solve it. If the structure is poorly understood, this may lead to a prohibitive amount of stored data. Some methods are becoming available for dealing with combinatorial explosive problems [13]; namely, the use of decision analysis approaches.

Heuristic methods differ from branch and bound algorithms in the sense that one is not sure of having the optimal solution at the end of the search; only a “good” solution. Heuristic searches have been tried on mathematical models as well as on real-life problems. Sometimes this is the only way to obtain a result. Heuristic criteria are also often used to accelerate the branch and bound procedure. Again the structure of the problem must be relatively well understood before applying this kind of criteria.

In the last few years, many applications have been reported in the literature. These can be divided into two categories.

The first deals with theoretical combinatorial problems that have been around for some time but for which good solution methods are yet to be

found. Among them are the traveling salesman problem [1, 3, 14, 15, 15a]; integer programming [5, 8, 9, 12, 16–19]; machine scheduling [20–22]; pseudo-Boolean programming [6, 10, 23]; set partitioning [24]; the knapsack problem [25]; nonconvex [26] or nonlinear problems [27].

The second category deals with more specific cases, where branch and bound techniques were applied successfully: synthesis of integrated process design [28]; computer aided synthesis of chemical plants [29]; ordering of recycle calculations [30]; project scheduling [31]; modular design [31a]; generation of NAND structures [32].

## REFERENCES

1. J. D. C. Little, K. G. Murty, D. W. Sweeney, and G. Karel, An algorithm for the traveling salesman problem. *Oper. Res.* **11**, 972 (1963).
2. C. Berge, "Théorie des Graphes et ses Applications." Dunod, Paris, 1967.
3. W. L. Eastman, A solution to the traveling salesman problem. *Amer. Summer Meeting Econometric Soc.* 1958.
4. F. S. Hillier and G. Lieberman, "Introduction to Operations Research." Holden-Day, San Francisco, California, 1967.
5. A. H. Land and A. Doig, Automatic method for solving discrete programming problems. *Econometrica* **28**, 497 (1960).
6. E. Balas, An additive algorithm for solving linear programs with 0-1 variables. *Oper. Res.* **13**, 517 (1965).
7. S. Ashour, Computational experience on 0-1 programming approach to various combinatorial problems. *Oper. Res. Jap.* **13**, 78 (1970).
8. M. L. Balinski, Integer programming, methods, uses, computations. *Management Sci.* **12**, 253 (1965).
9. R. Benayoun, B. Roy, and J. Tergny, De La Procedure S.E.P. au Programme Ophélie Mixte, *Metra* **9**, 141 (1970).
10. P. L. Hammer, A B-B-B method for linear and nonlinear bivalent programming. Rep. No. 48. *Oper. Res., Statist. and Econom.*, Technion, Haifa, Israel, 1969.
11. N. J. Nilsson, "Problem Solving in Artificial Intelligence." McGraw-Hill, New York, 1971.
12. J. S. Sirola and D. F. Rudd, Computer aided synthesis of chemical process designs. Chem. Eng. Dept., Univ. of Wisconsin, Madison, Wisconsin, 1971.
13. K. L. McRoberts, A search model for evaluating combinatorially explosive problems. *Oper. Res.* **19**, 1331 (1971).
14. M. Bellmore and G. L. Nemhauser, The traveling salesman, a review. *Oper. Res.* **16**, 538 (1968).
15. R. J. Freeman, Computational experience with Balas integer programming algorithm. Rep. P-3241. Rand Corp. Santa Monica, California, 1965.
- 15a. M. Held and R. M. Karp, The traveling salesman and minimum spanning tree. *Oper. Res.* **18**, 1138 (1970).

16. G. H. Bradley and P. N. Washi, An algorithm for integer linear programming, a combined algebraic and enumeration approach. Rep. No. 29. Administrative Sci., Yale Univ., New Haven, Connecticut, 1971.
17. A. M. Geoffrion, An improved implicit enumeration approach for integer programming. *Oper. Res.* **17**, 437 (1969).
18. M. M. Gutterman, Efficient implementation of a B and B algorithm. Oper. Res. Div., Standard Oil Co., Whiting, Indiana, 1969.
19. F. S. Hillier, Efficient heuristic procedure for integer linear programming with an interior. *Oper. Res.* **17**, 600 (1969).
20. E. Balas, Machine sequencing via disjunctive graphs: An implicit enumeration algorithm. *Oper. Res.* **17**, 941 (1969).
21. J. M. Charlton and C. C. Death, A method of solution for general machine scheduling problems. *Oper. Res.* **18**, 689 (1969).
22. M. Florian, P. Trepant, and G. McMahon, An implicit enumeration algorithm for the machine sequencing problem. *Management Sci.* **17**, B782 (1971).
- 22a. A. Kaufmann, "Introduction à la Combinatoire." Dunod, Paris, 1968.
23. E. Balas, Discrete programming by the filter method. *Oper. Res.* **15**, 915 (1967).
24. R. S. Garfinkel, and G. L. Nemhauser, The set partitioning problem: Set covering with equality constraints. *Oper. Res.* **17**, 848 (1969).
25. H. Greenberg and R. Hegerich, A branch search algorithm for the knapsack problem. *Management Sci.* **16**, 327 (1970).
26. M. Florian and P. Robillard, An implicit enumeration for the concave cost network flow problem. Publ. No. 12. Dept. Inform., Montréal, Canada, 1970.
27. E. L. Lawler, The quadratic assignment problem. *Management Sci.* **9**, 586 (1963).
- 27a. E. L. Lawler and D. E. Wood, Branch and bound methods: A survey. *Oper. Res.* **14**, 699 (1966).
28. K. F. Lee, A. H. Masso, and D. F. Rudd, Branch and bound synthesis of integrated process designs. *Ind. Eng. Chem. Fundam.* **9**, 1 (1970).
29. D. V. Steward, Partitioning and tearing systems of equations. *SIAM J. Numer. Anal.* **2**, 345 (1965).
30. W. Lee, J. Christensen, and D. Rudd, Design variable selection to simplify process calculations, *AIChE J.* **12**, 1104 (1966).
31. T. Mason and C. L. Moodie, A branch and bound algorithm for minimizing cost in project scheduling. *Management Sci.* **18**, B158, 1971.
- 31a. T. Shaftel, An integer approach to modular design. *Oper. Res.* **19**, 130 (1971).
32. E. S. Davidson, An algorithm for NAND decomposition under network constraints. *IEEE Trans. Comput.* **12**, 1098 (1969).

## CHAPTER 12

# PROCESS RELIABILITY ANALYSIS BY FLOW-GRAPH METHODS<sup>†</sup>

Reliability has become a key factor in the design and operation of today's large, complex, and expensive process systems [1-3]. It is no longer economically feasible to overdesign system facilities and to introduce excessive redundancy.

It is of great importance, therefore, both in system analysis and design, to have computer programs which efficiently predict the total reliability of complex systems, and which also give useful design information with respect to individual units.

In this chapter we develop a general computer algorithm to calculate the system reliability, given reliabilities of each unit in the system and relationships between the units in the form of a reliability graph. Besides giving the system reliability, the programs will calculate the sensitivity of the system reliability to individual system units.

<sup>†</sup> Principal authors: K. Inoue, Cullen College of Engineering, University of Houston, Houston, Texas (on leave from University of Kyoto, Kyoto, Japan) and S. Gandhi, Cullen College of Engineering, University of Houston, Houston, Texas.



The algorithm developed in this chapter is based on the path enumeration or tie set method [4]. Advantages and disadvantages of this approach have been discussed and compared with typical algorithms based on the state enumeration or event space method [4, 5].

### Module Representation of Reliability Graphs

Consider the modules in the complexly connected, directed reliability graph shown in Fig. 12.1. It is assumed that no more than one module between any two arbitrary nodes is permitted. A module is defined to be a

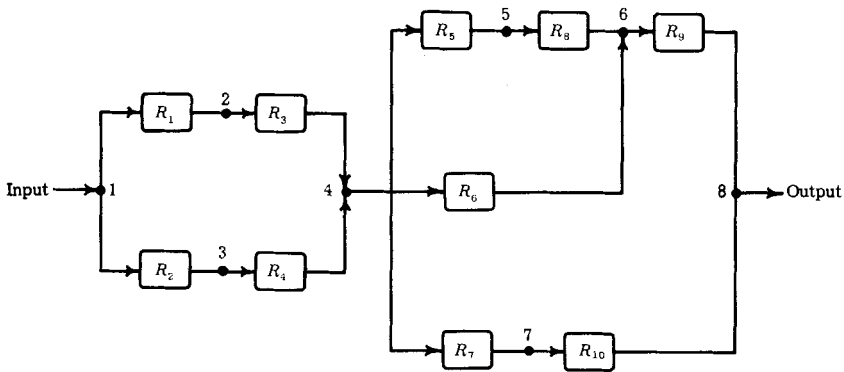


Fig. 12.1. Reliability graph.

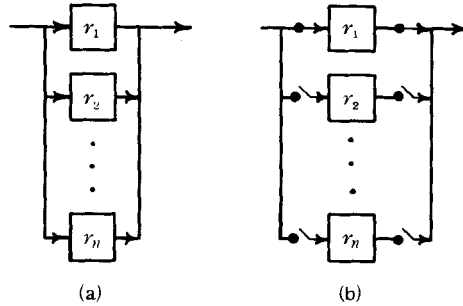
unit, or simply connected parallel units. A configuration permitted as a module is limited to either of the following:

#### I. SINGLE UNIT MODULE

A single unit module is a module which consists of a single unit. The module reliability is the reliability of the unit itself.

#### II. MULTI-UNIT MODULE

A module which consists of multiple parallel units is either an *active redundancy* module or a *standby redundancy* module as shown in Fig. 12.2.



**Fig. 12.2.** Multi-unit module. (a) Active redundancy module. (b) Standby redundancy module.

### 1. Active Redundancy Module

If units are connected actively in a module, the  $i$ th module reliability can be calculated by

$$R_i(r_{ij}) = 1 - \prod_{j=1}^{N_i} (1 - r_{ij}) \quad (12.1)$$

where  $r_{ij}$  is the reliability of the  $j$ th unit in the  $i$ th module, and  $N_i$  is the number of units in the  $i$ th module.

When the units are identical, that is  $r_{ij} = r_i$  for  $j = 1, 2, \dots, N_i$ , the module reliability is

$$R_i(N_i) = 1 - (1 - r_i)^{N_i} \quad (12.2)$$

### 2. Standby Redundancy Module

We distinguish between three standby redundancy modules. A unit whose failure probability in standby is zero is called a *cold standby*. Those whose failure probabilities are the same in standby as in service are called *hot standby* units. Other cases are called *warm*. It is assumed in this paper that all units in a standby redundancy module are identical.

**Cold Standby Module.** For the cold standby module, the module reliability is given by [6]

$$R_i(N_i) = \sum_{j=0}^{N_i-1} (r_i/j!)(-\ln r_i)^j \quad (12.3)$$

**Warm Standby Module.** For the warm standby case, the module reliability is given by [7]

$$R_i(N_i) = r^0 \left[ 1 + \sum_{k=1}^{N_i-1} (A_k/k!)(1 - r^1)^k \right] \quad (12.4)$$

where

$$A_k = \prod_{j=0}^{k-1} [j + (\ln r^0 / \ln r^1)] \quad (12.5)$$

and  $r^0$  and  $r^1$  are the reliability of a unit in service and in standby respectively.

*Hot Standby Module.* In the case of a hot standby module, the module reliability expression is the same as Eq. (12.2) of the active redundancy module.

### Principle of Path Enumeration and Sensitivity Calculation Methods

The traditional way of calculating the reliability of a system is to break the system into series and parallel subsystems, to calculate the subsystem reliabilities, and then to combine these through series or parallel formulations to yield the total system reliability. This method is not only unsuitable for computer programming, but also it cannot treat non-series-parallel systems such as those which have bridge type connections.

There are, however, two methods which can handle a general system configuration and are suitable for computer programming. One is the state enumeration method discussed recently by Brown [5]. The other is the path enumeration method. In this chapter, an algorithm based on the path enumeration method is developed. The algorithm proposed has several advantages over algorithms based on the state enumeration method when applied to complex systems; these will be discussed later.

The system reliability definition used in this chapter is the probability of the successful functioning of all of the modules in at least one path. A path is a group of branches which form a connection between input and output when traversed in a stated direction. A minimal path is a path which contains a minimum number of modules. If no node is traversed more than once in tracing a path, the path is minimal.

Let  $P_i$ ,  $i = 1, 2, \dots, M$ , denote the minimal path,  $M$  in number. The system reliability  $R$  can be expressed by

$$\begin{aligned} R &\equiv \text{probability \{at least one path is successful\}} \\ &= \Pr \left\{ \bigcup_{i=1}^M P_i \right\} \end{aligned} \quad (12.6)$$

where  $\cup$  denotes the union.

By use of the expansion rule for the probability of the union of  $M$  events [8], we have the formula:

$$\begin{aligned}
 R = & \sum_{i=1}^M \Pr\{P_i\} - \sum_{i=1}^M \sum_{j>i}^M \Pr\{P_i \cap P_j\} + \sum_{i=1}^M \sum_{j>i}^M \sum_{k>j}^M \Pr\{P_i \cap P_j \cap P_k\} \\
 & + \dots + (-1)^{M-1} \Pr\left\{\bigcap_{i=1}^M P_i\right\}
 \end{aligned} \quad (12.7)$$

where  $\cap$  denotes the intersection.

The total system reliability  $R$  is thus given in terms of module reliabilities by

$$\begin{aligned}
 R = & \sum_{i=1}^M \prod_{l \in P_i} R_l - \sum_{i=1}^M \sum_{j>i}^M \prod_{l \in P_i \cup P_j} R_l + \sum_{i=1}^M \sum_{j>i}^M \sum_{k>j}^M \prod_{l \in P_i \cup P_j \cup P_k} R_l \\
 & + \dots + (-1)^{M-1} \prod_{l \in \bigcup_{i=1}^M P_i} R_l
 \end{aligned} \quad (12.8)$$

where the members of the  $i$ th path, the union of the  $i$ th and the  $j$ th paths, etc. are denoted by  $l \in P_i$ ,  $l \in P_i \cup P_j$ , etc.

The total number of terms  $Z$  involved in Eq. (12.8) is given by

$$Z = 2^M - 1 \quad (12.9)$$

It should be noted in Eq. (12.8) that if, for example, the path  $P_1$  has modules  $R_1$ ,  $R_2$ , and  $R_3$ , and the path  $P_2$  has modules  $R_2$ ,  $R_3$ ,  $R_4$ , and  $R_5$ , then  $\Pr\{P_1 \cap P_2\}$  should be  $R_1 R_2 R_3 R_4 R_5$ , not  $R_1 R_2^2 R_3^2 R_4 R_5$ .

It is very interesting and important that the system reliability expression given by Eq. (12.8) is a bilinear function of each module's reliability. By use of this property, the sensitivity of the system reliability to the module  $R_i$  can be obtained by the simple rule

$$S_{R_i} = \partial R / \partial R_i = R(R_i = 1) - R(R_i = 0), \quad i = 1, 2, \dots, N \quad (12.10)$$

where  $N$  is the total number of modules. We call  $S_{R_i}$  a module sensitivity.

Two types of sensitivities are of interest: The first is the sensitivity of the system reliability to a unit reliability. A module sensitivity itself is of this type if the module is a single unit module. This type of sensitivity can also be defined for an active redundancy module with different units. This

sensitivity is easily calculated by

$$\begin{aligned}
 S_{r_{ij}} &= \frac{\partial R}{\partial r_{ij}} = \frac{\partial R}{\partial R_i} \frac{\partial R_i}{\partial r_{ij}} \\
 &= S_{R_i} \cdot \{R_i(r_{ij} = 1) - R_i(r_{ij} = 0)\} \\
 &= S_{R_i} \prod_{\substack{j=1 \\ j \neq i}}^{N_i} (1 - r_{ij})
 \end{aligned} \tag{12.11}$$

The second type of sensitivity is the sensitivity in terms of the number of units in a module. This type of sensitivity is defined for the active redundancy module or the standby modules with identical units. The sensitivity for the active redundancy module with identical units can be calculated by using the module sensitivity and Eq. (12.2) by

$$\begin{aligned}
 S_{N_i} &= \frac{\partial R}{\partial N_i} = \frac{\partial R}{\partial R_i} \frac{\partial R_i}{\partial N_i} \\
 &= S_{R_i} \cdot \{-(1 - r_i)^{N_i} \ln(1 - r_i)\}
 \end{aligned} \tag{12.12}$$

The sensitivity for the standby modules can be approximately calculated by

$$S_{N_i} \cong S_{R_i} \cdot \{R_i(N_i + 1) - R_i(N_i)\} \tag{12.13}$$

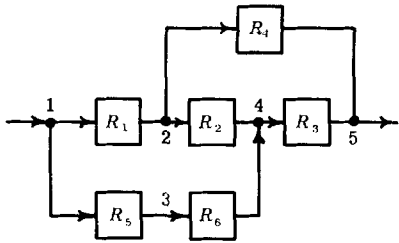
A large sensitivity value means that we may increase the system reliability very much by increasing the corresponding individual unit reliability or by increasing the number of units in the corresponding module. Therefore, since the sensitivity is a measure of system reliability improvement, it provides important information to system designers.

### Basic Algorithm

The total system reliability and the sensitivities can be obtained by the following six-step procedure:

1. Find all minimal paths using the reliability graph;
2. Find all the required unions of the paths;
3. Give each path union a reliability expression in terms of module reliabilities;

Fig. 12.3. Reliability graph.



4. Sum up all the reliability expressions obtained above according to Eq. (12.8);
5. Evaluate the system reliability by substituting numerical values of each module reliability; and
6. Evaluate the desired sensitivities by Eqs. (12.11), (12.12), or (12.13).

A brief description of each step is given and illustrated by way of the simple example shown in Fig. 12.3.

Path Finding Algorithm

We require a path finding routine which detects all of the minimal paths connecting input and output, given a system reliability graph. The path finding algorithm developed in Chapter 3 is ideally suited to this purpose. Given the reliability graph of Fig. 12.3, the routine detects three paths in the form shown in Table 12.1. By the nature of the algorithm, only the minimal paths are detected.

TABLE 12.1  
PATH DETECTION (IN NODE FORM)

Path	Trace
P1	1 → 2 → 4 → 5
P2	1 → 2 → 5
P3	1 → 3 → 4 → 5

Other path finding algorithms suitable for reliability studies and their FORTRAN listings are found in [9] and [10].

### System Reliability

Each detected path is converted into the binary form shown in Fig. 12.4. The locations containing 1's indicate the modules present in the path.  $N$  is the total number of modules. It is convenient for later manipula-

Sign	Module					
	1	2	3	4	...	$N$
—	1	0	0	1	...	1

Fig. 12.4. Binary representation of path or path union.

tions that the paths be given negative signs. In the case of Fig. 12.3, the three paths are converted into the binary forms shown in Fig. 12.5.

	Sign	Module					
		1	2	3	4	5	6
P1	—	1	1	1	0	0	0
P2	—	1	0	0	1	0	0
P3	—	0	0	1	0	1	1

Fig. 12.5. Binary representation of three paths.

The next step is to make the necessary unions of paths. This is easily done using a bitwise logical OR operation in the order of ascending path number. That is, (Path 1) OR (Path 2), (Path 1) OR (Path 3), ..., (Path 1) OR (Path  $M$ ), (Path 2) OR (Path 3), ..., (Path 2) OR (Path  $M$ ), ..., (Path  $M - 1$ ) OR (Path  $M$ ), which ends the 2-path unions. We then proceed to 3-path unions, ((Path 1) OR (Path 2)) OR (Path 3), ..., ((Path 1) OR (Path 2)) OR (Path  $M$ ), ..., ((Path  $M - 2$ ) OR (Path  $M - 1$ )) OR (Path  $M$ ). Similar procedures continue until we reach ((Path 1) OR (Path 2) OR ... OR (Path  $M - 1$ )) OR (Path  $M$ ). In the above operations, the sign must obey the ordinary multiplication rule.

The unions of paths for this example are shown in Fig. 12.6.

Each path or each path union corresponds to a term in Eq. (12.7) or (12.8) with an opposite sign. We now pick up every module that has a 1

	Sign	Module						
		1	2	3	4	5	6	
P1 OR P2	+	1	1	1	1	0	0	} Two-path unions
P1 OR P3	+	1	1	1	0	1	1	
P2 OR P3	+	1	0	1	1	1	1	
(P1 OR P2) OR P3	-	1	1	1	1	1	1	Three-path union

Fig. 12.6. Unions of paths.

in the corresponding location and form the products required by the reliability expression, Eq. (12.8). The collection of all the terms gives the required system reliability function with a negative sign.

For example, the system reliability function is given from Figs. 12.5 and 12.6 by

$$\begin{aligned}
 -R = & -(R_1R_2R_3 + R_1R_4 + R_3R_5R_6) + (R_1R_2R_3R_4 + R_1R_2R_3R_5R_6 \\
 & + R_1R_3R_4R_5R_6) - R_1R_2R_3R_4R_5R_6
 \end{aligned} \quad (12.14)$$

Given numerical values for the reliability of each module, we can obtain the system reliability. The module sensitivity  $S_{R_i}$  can be also easily calculated by evaluating the system reliability twice, once with  $R_i = 1$  and once with  $R_i = 0$ . The other sensitivities can also be calculated through the module sensitivities by Eqs. (12.11), (12.12), or (12.13).

### Comparison with State Enumeration Algorithm

An alternate method of calculating system reliability is the method of state enumeration. The algorithm for the state enumeration method can be briefly stated as follows [11]:

1. Find the all possible combinations of the states of the units (up or down).
2. For each combination which connects input and output, calculate the product of the unreliabilities of the down units and the reliabilities of the up units.



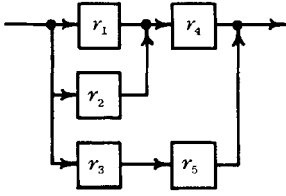


Fig. 12.7. Reliability graph (from [5]).

3. Sum up the products obtained in step 2. This gives the system reliability expression.

The algorithm can be easily computerized by using Boolean algebra [5]. As an example, consider the reliability graph shown in Figure 12.7. The algorithm can be understood by referring to Table 12.2. From the table, the system reliability function is given by

$$\begin{aligned}
 R = & (1 - r_1)(1 - r_2)r_3(1 - r_4)r_5 + (1 - r_1)(1 - r_2)r_3r_4r_5 \\
 & + (1 - r_1)r_2(1 - r_3)r_4(1 - r_5) + (1 - r_1)r_2(1 - r_3)r_4r_5 \\
 & + (1 - r_1)r_2r_3(1 - r_4)r_5 + (1 - r_1)r_2r_3r_4(1 - r_5) \\
 & + (1 - r_1)r_2r_3r_4r_5 + r_1(1 - r_2)(1 - r_3)r_4(1 - r_5) \\
 & + r_1(1 - r_2)(1 - r_3)r_4r_5 + r_1(1 - r_2)r_3(1 - r_4)r_5 \\
 & + r_1(1 - r_2)r_3r_4(1 - r_5) + r_1(1 - r_2)r_3r_4r_5 \\
 & + r_1r_2(1 - r_3)r_4(1 - r_5) + r_1r_2(1 - r_3)r_4r_5 + r_1r_2r_3(1 - r_4)r_5 \\
 & + r_1r_2r_3r_4(1 - r_5) + r_1r_2r_3r_4r_5 \quad (12.15)
 \end{aligned}$$

Paths and path unions required in the path enumeration algorithm are listed in Table 12.3. From Table 12.3, the system reliability function is easily derived as

$$R = r_1r_4 + r_2r_4 + r_3r_5 - r_1r_2r_4 - r_1r_3r_4r_5 - r_2r_3r_4r_5 + r_1r_2r_3r_4r_5 \quad (12.16)$$

Although the two expressions are different in form, it can be easily verified that both are equivalent.

We can conclude from the above comparison and from more general consideration that the advantages of the path enumeration method are

1. Steps to reach the system reliability function are much less if the number of paths is fairly small and/or if the number of modules is fairly large.
2. The number of terms in the system reliability function are fewer.
3. Any term in the system reliability function contains equal or fewer multipliers.

TABLE 12.2  
ALL COMBINATION OF THE STATES

State number	Binary number					Giving a path?
	$r_1$	$r_2$	$r_3$	$r_4$	$r_5$	
0	0	0	0	0	0	No
1	0	0	0	0	1	No
2	0	0	0	1	0	No
3	0	0	0	1	1	No
4	0	0	1	0	0	No
5	0	0	1	0	1	Yes
6	0	0	1	1	0	No
7	0	0	1	1	1	Yes
8	0	1	0	0	0	No
9	0	1	0	0	1	No
10	0	1	0	1	0	Yes
11	0	1	0	1	1	Yes
12	0	1	1	0	0	No
13	0	1	1	0	1	Yes
14	0	1	1	1	0	Yes
15	0	1	1	1	1	Yes
16	1	0	0	0	0	No
17	1	0	0	0	1	No
18	1	0	0	1	0	Yes
19	1	0	0	1	1	Yes
20	1	0	1	0	0	No
21	1	0	1	0	1	Yes
22	1	0	1	1	0	Yes
23	1	0	1	1	1	Yes
24	1	1	0	0	0	No
25	1	1	0	0	1	No
26	1	1	0	1	0	Yes
27	1	1	0	1	1	Yes
28	1	1	1	0	0	No
29	1	1	1	0	1	Yes
30	1	1	1	1	0	Yes
31	1	1	1	1	1	Yes

TABLE 12.3  
PATHS AND PATH UNIONS

Paths and path unions	Sign	Module				
		$r_1$	$r_2$	$r_3$	$r_4$	$r_5$
P1	—	1	0	0	1	0
P2	—	0	1	0	1	0
P3	—	0	0	1	0	1
P1 OR P2	+	1	1	0	1	0
P1 OR P3	+	1	0	1	1	1
P2 OR P3	+	0	1	1	1	1
(P1 OR P2) OR P3	—	1	1	1	1	1

In the algorithm developed in this paper a reduction in the number of paths can be achieved by introducing the module structure. The units  $r_1$  and  $r_2$  are combined and treated as the active redundancy module  $R_1$  at the first stage as shown in Fig. 12.8. Then the path enumeration algorithm

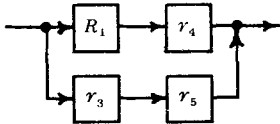


Fig. 12.8. Module representation of Fig. 12.7.

is applied. Paths and path unions are shown in Table 12.4. The system reliability is given in the form

$$R = R_1r_4 + r_3r_5 - R_1r_3r_4r_5 \tag{12.17}$$

TABLE 12.4  
PATHS AND PATH UNION

Paths and path union	Sign	Module			
		$R_1$	$r_3$	$r_4$	$r_5$
P1	—	1	0	1	0
P2	—	0	1	0	1
P1 OR P2	+	1	1	1	1

where

$$R_1 = 1 - (1 - r_1)(1 - r_2) \quad (12.18)$$

In this way, the previously stated advantages of the path enumeration method can be enhanced by introducing the module structure. Since it is customary to use parallel redundancy configurations to increase the reliability of a weak unit, the module concept can reduce the number of paths, and hence the computing time, as is now shown by more complicated examples.

### Examples

At first, consider the fairly complicated reliability graph shown in Figure 12.9. The reliability of each unit is given in Table 12.5. In this original graph, there are 16 units and 55 paths. The number of paths and path unions required for calculating the system reliability, therefore, will be  $2^{55} - 1$  from Eq. (12.9), which is far in excess of  $10^{16}$ . The application of the path enumeration method to the original graph is thus very time consuming, if not impossible.

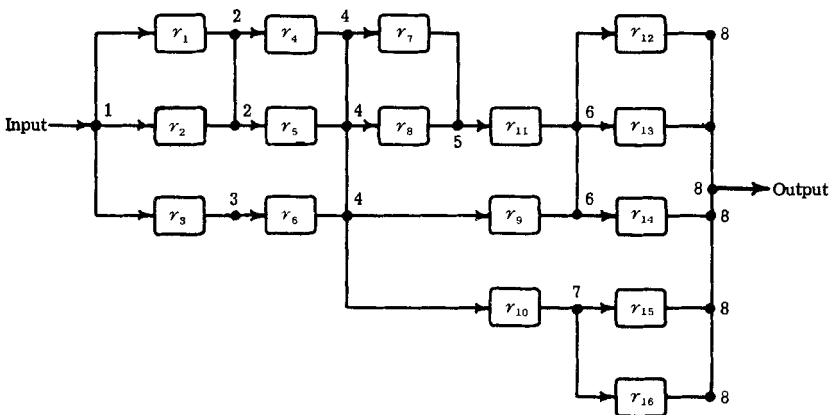


Fig. 12.9. Reliability graph with 55 paths (from [9] with slight modification).

By using the module concept, the graph is automatically converted to Fig. 12.1, where  $R_1$ ,  $R_3$ ,  $R_5$ ,  $R_9$ ; and  $R_{10}$  are actively connected multi-unit modules which consist of  $r_1$  and  $r_2$ ,  $r_4$  and  $r_5$ ,  $r_7$  and  $r_8$ ,  $r_{12}$ ,  $r_{13}$ ; and  $r_{14}$ , and  $r_{15}$  and  $r_{16}$ , respectively;  $R_2$ ,  $R_4$ ,  $R_6$ ,  $R_7$ , and  $R_8$  are single unit modules consisting of  $r_3$ ,  $r_6$ ,  $r_9$ ,  $r_{10}$ , and  $r_{11}$ , respectively. This is shown in Table 12.6.

TABLE 12.5  
UNIT RELIABILITIES

Unit	Reliability
$r_1$	0.80
$r_2$	0.70
$r_3$	0.90
$r_4$	0.75
$r_5$	0.85
$r_6$	0.87
$r_7$	0.82
$r_8$	0.62
$r_9$	0.88
$r_{10}$	0.75
$r_{11}$	0.89
$r_{12}$	0.85
$r_{13}$	0.75
$r_{14}$	0.65
$r_{15}$	0.70
$r_{16}$	0.90

TABLE 12.6  
MODULE REPRESENTATION

Module	Branch	Unit reliability	Number of units
1	$1 \rightarrow 2$	0.800000	2
		0.700000	
2	$1 \rightarrow 3$	0.900000	1
3	$2 \rightarrow 4$	0.750000	2
		0.850000	
4	$3 \rightarrow 4$	0.870000	1
5	$4 \rightarrow 5$	0.820000	2
		0.620000	
6	$4 \rightarrow 6$	0.880000	1
7	$4 \rightarrow 7$	0.750000	1
8	$5 \rightarrow 6$	0.890000	1
9	$6 \rightarrow 8$	0.850000	3
		0.750000	
		0.650000	
10	$7 \rightarrow 8$	0.700000	2
		0.900000	

The converted graph has only six paths as shown in Table 12.7. The binary representation of the six paths is shown in Table 12.8.

Since the number of paths is greatly reduced, our algorithm can be applied easily and efficiently to produce the system reliability value of 0.97043 in a few seconds.

TABLE 12.7  
PATH DETECTION (IN NODE FORM)

Path	Trace
1	1 → 2 → 4 → 6 → 8
2	1 → 3 → 4 → 6 → 8
3	1 → 2 → 4 → 5 → 6 → 8
4	1 → 3 → 4 → 5 → 6 → 8
5	1 → 2 → 4 → 7 → 8
6	1 → 3 → 4 → 7 → 8

TABLE 12.8  
BINARY REPRESENTATION OF PATHS

Path	Module									
	1	2	3	4	5	6	7	8	9	10
1	1	0	1	0	0	1	0	0	1	0
2	0	1	0	1	0	1	0	0	1	0
3	1	0	1	0	1	0	0	1	1	0
4	0	1	0	1	1	0	0	1	1	0
5	1	0	1	0	0	0	1	0	0	1
6	0	1	0	1	0	0	1	0	0	1

The sensitivities are listed in Table 12.9, where attention is focused on the first type of sensitivity,  $\partial R/\partial r_{ij}$ , given by Eq. (12.11). The table shows that the module  $R_4$ , that is, unit  $r_6$ , has the largest sensitivity value of 0.085 and the first unit of the module  $R_{10}$ , unit  $r_{15}$ , gives the smallest. It is, therefore, more efficient to increase the unit reliability of  $r_6$  in order to increase the total reliability of the system. Moreover the sensitivity tells us that the increase in the total reliability will amount to 0.0085 when the increase in the unit reliability of  $r_6$  is 0.1.

TABLE 12.9  
SENSITIVITIES  $\partial R/\partial r_{ij}$

Module	Module reliability	Module sensitivity $\partial R/\partial R_i$	Unit sensitivity $\partial R_i/\partial r_{ij}$	System sensitivity $\partial R/\partial r_{ij}$
1	0.940000	0.206964	0.300000	0.620891E-01
			0.200000	0.413928E-01
2	0.900000	0.821141E-01	1.000000	0.821141E-01
3	0.962500	0.202126	0.150000	0.303189E-01
			0.250000	0.505314E-01
4	0.870000	0.849457E-01	1.000000	0.849456E-01
5	0.931600	0.281274E-01	0.380000	0.106884E-01
			0.180000	0.506293E-02
6	0.880000	0.450027E-01	1.000000	0.450027E-01
7	0.750000	0.316913E-01	1.000000	0.316913E-01
8	0.890000	0.294421E-01	1.000000	0.294421E-01
9	0.986875	0.261395	0.875000E-01	0.228721E-01
			0.525000E-01	0.137233E-01
			0.375000E-01	0.980232E-02
10	0.970000	0.245036E-01	0.100000	0.245036E-02
			0.300000	0.735107E-02

As the second example, we consider a more realistic system. Figure 12.10 shows a boiler DDC safety system (pressure safety system). Reliabilities of each unit are listed in Table 12.10. The original graph has 26 paths. After being converted to the module configuration shown in Fig. 12.11 and Table 12.11, the number of paths is reduced to only nine as shown in Tables 12.12 and 12.13.

The total reliability of the system is computed to be 0.99927. The sensitivities are listed in Table 12.14. In this case, the sensitivities are evaluated with respect to the number of units in a module, that is,  $\partial R/\partial N_i$  defined by Eq. (12.12). The table shows that the module  $R_4$  has the largest sensitivity and the module  $R_5$  has the smallest sensitivity. Adding one more unit (Converter 1) to the module  $R_5$  is of no use, as the increase in the total reliability will be only 0.0000009.

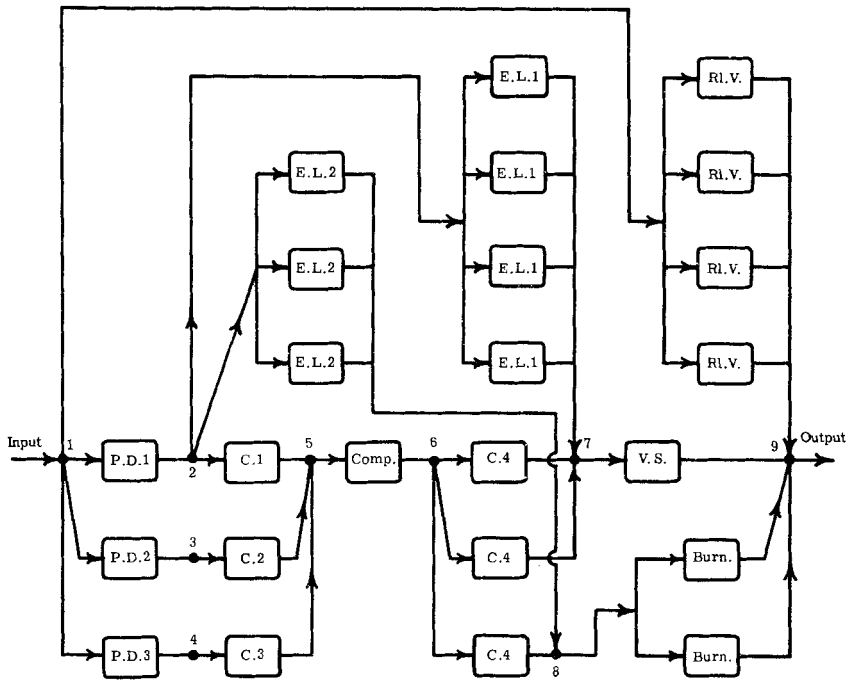


Fig. 12.10. Reliability graph of a boiler DDC safety system (from [11] with slight modification).

TABLE 12.10  
UNIT RELIABILITIES

Unit	Reliability
Pressure detector 1 (P.D.1)	0.68
Pressure detector 2 (P.D.2)	0.75
Pressure detector 3 (P.D.3)	0.75
Converter 1 (C.1)	0.70
Converter 2 (C.2)	0.70
Converter 3 (C.3)	0.70
Converter 4 (C.4)	0.70
Converter 5 (C.5)	0.70
Computer (Comp.)	0.70
Valve servo (V.S.)	0.75
Burner (Burn.)	0.68
Relief valve (Rl.V.)	0.75
Emergency line 1 (E.L.1)	0.68
Emergency line 2 (E.L.2)	0.68



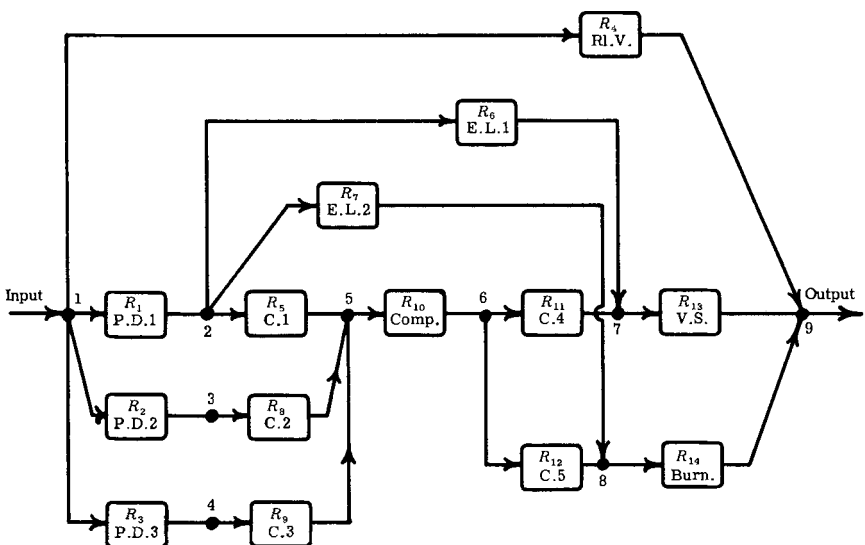


Fig. 12.11. Module representation of a boiler DDC safety system.

TABLE 12.11  
MODULE REPRESENTATION

Module	Branch	Unit reliability	Number of units
1	1 → 2	0.680000	1
2	1 → 3	0.750000	1
3	1 → 4	0.750000	1
4	1 → 9	0.750000	4
5	2 → 5	0.700000	1
6	2 → 7	0.680000	4
7	2 → 8	0.680000	3
8	3 → 5	0.700000	1
9	4 → 5	0.700000	1
10	5 → 6	0.700000	1
11	6 → 7	0.700000	2
12	6 → 8	0.700000	1
13	7 → 9	0.750000	1
14	8 → 9	0.680000	2

TABLE 12.12  
PATH DETECTION (IN NODE FORM)

Path	Trace
1	1 → 9
2	1 → 2 → 7 → 9
3	1 → 2 → 5 → 6 → 7 → 9
4	1 → 3 → 5 → 6 → 7 → 9
5	1 → 4 → 5 → 6 → 7 → 9
6	1 → 2 → 8 → 9
7	1 → 2 → 5 → 6 → 8 → 9
8	1 → 3 → 5 → 6 → 8 → 9
9	1 → 4 → 5 → 6 → 8 → 9

TABLE 12.13  
BINARY REPRESENTATION OF PATHS

Path	Module													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	1	0	0	0	0	0	0	1	0
3	1	0	0	0	1	0	0	0	0	1	1	0	1	0
4	0	1	0	0	0	0	0	1	0	1	1	0	1	0
5	0	0	1	0	0	0	0	0	1	1	1	0	1	0
6	1	0	0	0	0	0	1	0	0	0	0	0	0	1
7	1	0	0	0	1	0	0	0	0	1	0	1	0	1
8	0	1	0	0	0	0	0	1	0	1	0	1	0	1
9	0	0	1	0	0	0	0	0	1	1	0	1	0	1

TABLE 12.14  
SENSITIVITIES  $\partial R/\partial N_i$

Module	Module reliability	Module sensitivity $\partial R/\partial R_i$	Unit sensitivity $\partial R_i/\partial N_i$	System sensitivity $\partial R/\partial N_i$
1	0.680000	0.192148E-02	0.364619	0.700609E-03
2	0.750000	0.257737E-03	0.346574	0.893248E-04
3	0.750000	0.257737E-03	0.346574	0.893248E-04
4	0.996094	0.187413	0.541521E-02	0.101488E-02
5	0.700000	0.256121E-05	0.361192	0.925088E-06
6	0.989514	0.104235E-03	0.119478E-01	0.124538E-05
7	0.967232	0.330605E-03	0.373370E-01	0.123438E-04
8	0.700000	0.276147E-03	0.361192	0.997419E-04
9	0.700000	0.276147E-03	0.361192	0.997419E-04
10	0.700000	0.868859E-03	0.361192	0.313825E-03
11	0.910000	0.190398E-03	0.108357	0.206310E-04
12	0.700000	0.205885E-03	0.361192	0.743638E-04
13	0.750000	0.542155E-03	0.346574	0.187897E-03
14	0.897600	0.811069E-03	0.116678	0.946339E-04

### Extension to System MTBF Calculation

When a unit reliability is a function of time, the most important parameter of the system reliability is the mean time between failures (MTBF). The system MTBF is defined as

$$\bar{M} = \int_0^{\infty} R(t) dt \quad (12.19)$$

The sensitivity of the MTBF to a parameter  $\lambda_{ij}$  in the  $j$ th unit of the  $i$ th module, and to the number of units in the  $i$ th module is given by

$$\frac{\partial \bar{M}}{\partial \lambda_{ij}} = \int_0^{\infty} \frac{\partial R}{\partial R_i} \frac{\partial R_i}{\partial r_{ij}} \frac{\partial r_{ij}}{\partial \lambda_{ij}} dt \quad (12.20)$$

and

$$\frac{\partial \bar{M}}{\partial N_i} = \int_0^{\infty} \frac{\partial R}{\partial R_i} \frac{\partial R_i}{\partial N_i} dt \quad (12.21)$$

Therefore, by addition of an integration routine to our program, we can easily evaluate the system MTBF and its sensitivities. The present

program system contains a numerical integration routine based on Simpson's rule.

The type of reliability function of each unit is limited to the exponential type with constant failure rate. That is, (1) for an active redundancy module

$$R_i(r_{ij}) = 1 - \prod_{j=1}^{N_i} [1 - \exp(-\lambda_{ij}t)] \quad (12.22)$$

(2) for a cold standby module

$$R_i(N_i) = \sum_{j=0}^{N_i-1} \frac{\exp(-\lambda_i t)}{j!} (\lambda_i t)^j \quad (12.23)$$

and (3) for a warm standby module

$$R_i(N_i) = \exp(-\lambda_{0i}t) \left\{ 1 + \sum_{k=1}^{N_i-1} \frac{A_k}{k!} [1 - \exp(-\lambda_{1i}t)]^k \right\} \quad (12.24)$$

where

$$A_k = \prod_{j=0}^{k-1} (j + \lambda_{0i}/\lambda_{ij}) \quad (12.25)$$

In Eqs. (12.22) and (12.23),  $\lambda_{ij}$  and  $\lambda_i$  are constant failure rates.  $\lambda_{0i}$  and  $\lambda_{1i}$  of Eqs. (12.24) and (12.25) are constant failure rates in service and in standby respectively.

The MTBF calculation routine developed here is more flexible and easier to apply to more complicated systems than the RELCOMP routine of Fleming [12] which is only applicable to series-parallel configurations.

## Conclusions

The procedures presented have been programmed in FORTRAN IV for the IBM 360-44. Because of the incapability of the FORTRAN IV language to deal with bitwise operations, slight modifications of the form of the binary path expression shown in Fig. 12.4 were necessary.

The proposed algorithm is more powerful and efficient for deriving and evaluating the total system reliability, the sensitivities, and the MTBF of complex systems than other algorithms based on the state enumeration method, since systems are often complex enough to have many units, but not so complicated as to have many paths in the module representation.

## REFERENCES

1. P. S. Ufford, Equipment reliability analysis for large plants. *Chem. Eng. Prog.* **68**, 47-49 (1972).
2. W. E. McFatter, Reliability experiences in a large refinery. *Chem. Eng. Prog.* **68**, 52-55 (1972).
3. K. E. Coulter and V. S. Morello, Improving onstream time in process plants. *Chem. Eng. Prog.* **68**, 56-59 (1972).
4. M. L. Shooman, "Probabilistic Reliability: An Engineering Approach." McGraw-Hill, New York, 1968.
5. D. B. Brown, A computerized algorithm for determining the reliability of redundant configurations. *IEEE Trans. Rel.* **R-20**, 102-107 (1971).
6. C. J. Benning, Reliability prediction formulas for standby redundant structures. *IEEE Trans. Rel.* **R-16**, 136-137 (1967).
7. A. M. Polovko, "Fundamentals of Reliability Theory." Academic Press, New York, 1968.
8. W. Feller, "An Introduction to Probability Theory and Its Applications," Vol. 1. Wiley, New York, 1957.
9. A. C. Nelson, Jr., J. R. Batts, and R. L. Beadles, A computer program for approximating system reliability. *IEEE Trans. Rel.* **R-19**, 61-65 (1970).
10. J. R. Batts, Computer program for approximating system reliability, Part II. *IEEE Trans. Rel.* **R-20**, 85-87 (1971).
11. T. Terano, Y. Murayama, and K. Kurosu, Optimum design of safety systems. *IFAC Kyoto Symposium, August 1970*. Preprints, pp. 52-57.
12. J. L. Fleming, RELCOMP: a computer program for calculating system reliability and MTBF. *IEEE Trans. Rel.* **R-20**, 102-107 (1971).

## APPENDIX A

# MATRIX THEORY

### Matrices

A *matrix*  $\mathbf{A}$  is a rectangular array of numbers. The numbers in the array,  $a_{ij}$ , are the elements of the matrix.

$$\mathbf{A} = (a_{ij}) = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

The matrix  $\mathbf{A}$  is of dimension  $m \times n$  since it has  $m$  rows and  $n$  columns. Several special matrices are of interest.

- For a *square matrix*,  $m = n$ .
- A *diagonal matrix* is a square matrix having elements only along the diagonal,  $a_{ij} = 0$  for  $i \neq j$ .
- An *identity matrix*  $\mathbf{I}$  is a diagonal matrix where  $a_{ij} = 1$  for  $i = j$ .

d. *Matrix transpose*: An  $n \times m$  matrix obtained by interchanging the rows and columns of an  $m \times n$  matrix is the transpose of  $\mathbf{A}$  and will be denoted by  $\mathbf{A}^T$ , where  $\mathbf{A} = (a_{ij})$ ,  $\mathbf{A}^T = (a_{ji})$ .

e. A *symmetric matrix* is a square matrix such that  $\mathbf{A}^T = \mathbf{A}$ .

f. A *skew-symmetric matrix* is a matrix  $\mathbf{A}$  for which  $\mathbf{A} = -\mathbf{A}^T$ .

g. A *singular matrix* is a square matrix whose determinant is zero.

A number of matrix manipulations are of importance.

a. *Addition*: Two matrices of equal dimensions may be added, element by element

$$\mathbf{A} + \mathbf{B} = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots & a_{2n} + b_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \cdots & a_{mn} + b_{mn} \end{pmatrix}$$

In general, for matrices  $\mathbf{A}$ ,  $\mathbf{B}$ , and  $\mathbf{C}$  (of the same dimensions),

$$(1) \quad (\mathbf{A} + \mathbf{B}) + \mathbf{C} = \mathbf{A} + (\mathbf{B} + \mathbf{C})$$

$$(2) \quad \mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A}$$

$$(3) \quad \mathbf{A} + \mathbf{0} = \mathbf{0} + \mathbf{A} = \mathbf{A}$$

b. *Multiplication by a scalar*: to multiply matrix  $\mathbf{A}$  by a scalar  $t$ , we multiply every element in  $\mathbf{A}$  by  $t$ .

$$\mathbf{B} = t\mathbf{A} = \begin{pmatrix} ta_{11} & ta_{12} & \cdots & ta_{1n} \\ ta_{21} & ta_{22} & \cdots & ta_{2n} \\ \vdots & \vdots & & \vdots \\ ta_{m1} & ta_{m2} & \cdots & ta_{mn} \end{pmatrix}$$

In general, for matrices of equal dimensions

$$(1) \quad t_1 t_2 \mathbf{A} = t_1 (t_2 \mathbf{A})$$

$$(2) \quad t_1 (\mathbf{A} + \mathbf{B}) = t_1 \mathbf{A} + t_1 \mathbf{B}$$

$$(3) \quad (t_1 + t_2)(\mathbf{A}) = t_1 \mathbf{A} + t_2 \mathbf{A}$$

$$(4) \quad 1\mathbf{A} = \mathbf{A} \quad \text{and} \quad 0\mathbf{A} = \mathbf{0}$$

$$(5) \quad \mathbf{A} + (-\mathbf{A}) = (-\mathbf{A}) + \mathbf{A} = \mathbf{0}$$

c. *Matrix multiplication*: Only matrices of equal “inner dimensions” can be multiplied. By the product  $\mathbf{AB}$  (in that order) of the  $m \times p$  matrix,  $\mathbf{A} = (a_{ij})$  and the  $p \times n$  matrix  $\mathbf{B} = (b_{ij})$  is meant the  $m \times n$  matrix  $\mathbf{C} = (C_{ij})$  where

$$C_{ij} = \sum_{k=1}^p a_{ik}b_{kj}, \quad \begin{matrix} i = 1, 2, \dots, m \\ j = 1, 2, \dots, n \end{matrix}$$

For matrices  $\mathbf{A}$ ,  $\mathbf{B}$ , and  $\mathbf{C}$  of proper dimensions, and scalar  $t$

- (1)  $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$
- (2)  $\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$
- (3)  $(\mathbf{B} + \mathbf{C})\mathbf{A} = \mathbf{BA} + \mathbf{CA}$
- (4)  $t(\mathbf{AB}) = (t\mathbf{A})(\mathbf{B}) = \mathbf{A}(t\mathbf{B})$

d. *Matrix inverse*: If  $\mathbf{A}$  and  $\mathbf{B}$  are nonsingular square matrices such that

$$\mathbf{AB} = \mathbf{BA} = \mathbf{I}$$

where  $\mathbf{I}$  is the identity matrix, then  $\mathbf{B}$  is called the inverse of  $\mathbf{A}$ . The inverse of  $\mathbf{A}$  is written as  $\mathbf{A}^{-1}$ .

e. *Partitioning of matrices*: If we cross out all but  $k$  rows and  $s$  columns of an  $m \times n$  matrix  $\mathbf{A}$ , the resulting matrix is called a *submatrix* of  $\mathbf{A}$ , and  $\mathbf{A}$  is said to have been partitioned. Imagine  $\mathbf{A}$  to be partitioned by the dashed lines.

$$\mathbf{A} = \left( \begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \\ a_{51} & a_{52} & a_{53} & a_{54} \end{array} \right)$$

Let

$$\begin{aligned} \mathbf{A}_{11} &= \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}, & \mathbf{A}_{12} &= \begin{pmatrix} a_{14} \\ a_{24} \end{pmatrix} \\ \mathbf{A}_{21} &= \begin{pmatrix} a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \\ a_{51} & a_{52} & a_{53} \end{pmatrix}, & \mathbf{A}_{22} &= \begin{pmatrix} a_{34} \\ a_{44} \\ a_{54} \end{pmatrix} \end{aligned}$$

$\mathbf{A}$  can now be written as

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix}$$



If a matrix  $\mathbf{B}$  is similarly partitioned, then the sum is

$$\mathbf{A} + \mathbf{B} = \begin{pmatrix} \mathbf{A}_{11} + \mathbf{B}_{11} & \mathbf{A}_{12} + \mathbf{B}_{12} \\ \mathbf{A}_{21} + \mathbf{B}_{21} & \mathbf{A}_{22} + \mathbf{B}_{22} \end{pmatrix}$$

Also, providing the submatrices are conformable,

$$\mathbf{C} = \mathbf{AB} = \begin{pmatrix} \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{12}\mathbf{B}_{21} & \mathbf{A}_{11}\mathbf{B}_{12} + \mathbf{A}_{12}\mathbf{B}_{22} \\ \mathbf{A}_{21}\mathbf{B}_{11} + \mathbf{A}_{22}\mathbf{B}_{21} & \mathbf{A}_{21}\mathbf{B}_{12} + \mathbf{A}_{22}\mathbf{B}_{22} \end{pmatrix}$$

To carry out this operation, it is necessary that the partitioning of the columns in the premultiplier be the same as the rows of the postmultiplier.

## Vectors

A matrix that has a single column or a single row is a vector. A column vector  $\mathbf{X}$  has the dimensions  $(m \times 1)$ . A row vector  $\mathbf{X}^T$  is a  $(1 \times m)$  matrix.

Vectors may be multiplied by matrices or other vectors providing conformability is maintained.

The *inner* (dot) *product* of two vectors  $\mathbf{X}$  and  $\mathbf{Y}$  is

$$\mathbf{X} \cdot \mathbf{Y} = \mathbf{X}^T \mathbf{Y} = (x_1 y_1 + x_2 y_2 + \cdots + x_n y_n)$$

## Independence, Orthogonality, and Rank

A set of vectors  $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n$  (each of  $m$  components) are linearly dependent if a set of scalars,  $t_1, t_2, \dots, t_n$ , not all zero, exist such that

$$t_1 \mathbf{X}_1 + t_2 \mathbf{X}_2 + \cdots + t_n \mathbf{X}_n = \mathbf{0}$$

If such a set does not exist, the vectors are linearly independent. In addition, a pair of vectors are orthogonal if

$$\mathbf{X}_1 \cdot \mathbf{X}_2 = \mathbf{X}_1^T \mathbf{X}_2 = 0$$

The rank (or more precisely the columnar rank) of an  $m \times n$  matrix  $\mathbf{A}$  is the maximum number of independent column vectors in  $\mathbf{A}$ . For a set of  $m$  vectors of  $n$  components, the rank can be at most the smaller of  $n$  or  $m$ . For a square matrix, the rank of the column vectors must equal the rank of the row vectors.

### Determinants

The determinant of an  $n$ th-order square matrix  $\mathbf{A} = (a_{ij})$ , written as  $|\mathbf{A}|$ , is the scalar number computed from the sum involving the  $n!$  elements in  $\mathbf{A}$

$$|\mathbf{A}| = \sum_{n!} (\pm)(a_{i_1 1} a_{i_2 2} a_{i_3 3}, \dots, a_{i_n n})$$

the sum being taken over all permutations  $i_1, i_2, \dots, i_n$  of  $1, 2, \dots, n$ . A term is assigned a plus or minus sign depending on whether the permutation  $i_1, i_2, \dots, i_n$  is even or odd. The permutation of two symbols of a permutation changes the sign of the permutation. There are  $n!$  terms in the summation. Only one element from each row and column of  $\mathbf{A}$  appears in every term of the summation.

Some properties of determinants are: (a) an interchange of either two rows or two columns of a determinant  $|\mathbf{A}|$  changes its sign; (b) for a square matrix  $\mathbf{A}$ ,  $|\mathbf{A}| = |\mathbf{A}^T|$ .

### Cofactors, Adjoint, Minors

The cofactor  $\mathbf{A}_{ij}$  of the element  $a_{ij}$  of any square matrix  $\mathbf{A}$  is  $(-1)^{i+j}$  times the determinant of the submatrix obtained from  $\mathbf{A}$  by deleting row  $i$  and column  $j$ .

$|\mathbf{A}_{ij}|$  is the *minor* of  $a_{ij}$  and the evaluation of a determinant can be made by an *expansion by minors*:

$$|\mathbf{A}| = \sum_{j=1}^n (-1)^{i+j} a_{ij} |\mathbf{A}_{ij}|$$

For example,

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

The adjoint  $(\text{adj})\mathbf{A}$  of  $\mathbf{A}$  is the matrix  $\mathbf{B} = (b_{ij})^T$  where

$$b_{ij} = (-1)^{i+j} |\mathbf{A}_{ij}|$$

### Systems of Linear Equations

A set of  $m$  linear equations in  $n$  unknowns has the form

$$\begin{array}{ccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & = & b_m \end{array}$$

In matrix notation, this may be expressed as

$$\mathbf{A}\mathbf{X} = \mathbf{B}$$

where  $\mathbf{A}$  is  $m \times n$ ,  $\mathbf{X}$  is  $n \times 1$ , and  $\mathbf{B}$  is  $m \times 1$ .

If  $\mathbf{B}$  is the zero vector, the equations are *homogeneous* and have the solution  $\mathbf{X} = 0$ . If the rank of the  $n$  vectors of the  $\mathbf{A}$  matrix is the same as the rank of the *augmented matrix* formed by attaching the  $\mathbf{B}$  vector as a column to the  $\mathbf{A}$  matrix, the equations are *consistent*. Otherwise they are inconsistent and have no solution. If the equations are consistent, and the  $\mathbf{A}$  matrix is square with a rank of  $n$  (equal to the number of unknowns), a unique solution can be obtained.

One method of obtaining a solution is by *Cramers rule*. If  $|\mathbf{A}| \neq 0$ , then

$$x_k = |\mathbf{A}(k)| / |\mathbf{A}|$$

where  $|\mathbf{A}(k)|$  is the matrix obtained from  $\mathbf{A}$  by replacing the  $k$ th column by the vector of constants  $\mathbf{B}$ .

From the computational standpoint, the most efficient methods for solving linear equations are based on Gaussian elimination. This involves forming the augmented coefficient matrix

$$\begin{array}{cccccc} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}$$

and transforming it by repeated addition and subtraction of the rows

(multiplied by suitable constants) to the diagonal matrix

$$\begin{array}{cccccc} 1 & 0 & \dots & 0 & b_1^n \\ 0 & 1 & & 0 & b_2^n \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & 1 & b_n^n \end{array}$$

The  $b_1^n \dots b_n^n$ 's then become the values of the unknowns  $x_1 \dots x_n$ .

The inverse of a matrix may be found in an analogous fashion since  $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$ . Thus  $\mathbf{A}^{-1}$  becomes the "unknown," and the identity matrix is appended to the matrix of coefficients in place of the vector of constants  $\mathbf{B}$ .

This page intentionally left blank

## APPENDIX B

### LINEARIZATION OF EQUATIONS

In a linear algebraic equation, all of the variables occur to the first power, and there are no terms involving products of variables. A linear ordinary differential is one where the dependent variable and its derivatives occur only to the first power, and no terms in the equation contains either a product of the dependent variable and any of its derivatives or a product of the derivatives of the dependent variable. The coefficients in a linear differential equation may be constants or functions of the independent variable.

Since the methods of flow-graph analysis can be applied only to linear algebraic and linear differential equations, it is of considerable importance that we examine methods of transforming nonlinear equations into linear ones.

The most common way of moving from a point  $f(x_0)$  in a solution of an algebraic or differential equation to a following point  $f(x)$  is to make use of Taylor's series. For a function of one variable, the first three terms in the series are

$$\begin{aligned} z = f(x) = & f(x_0) + \left. \frac{df}{dx} \right|_{x_0} (x - x_0) + \frac{1}{2} \left. \frac{d^2f}{dx^2} \right|_{x_0} (x - x_0)^2 \\ & + \frac{1}{6} \left. \frac{d^3f}{dx^3} \right|_{x_0} (x - x_0)^3 + \dots \end{aligned} \quad (\text{B.1})$$

The derivatives can be evaluated by differentiating the original equation and substituting values of  $x_0$ . If only the first two terms of the right-hand side are retained, we obtain the linear expression

$$z' = f(x)' = a + bx \quad (\text{B.2})$$

where the prime denotes an approximation. This approximation is valid and useful only if the series converges rapidly, i.e.,  $(x - x_0)$  is small. For a function of two variables, retaining only the first two terms

$$z' = f(x, y)' \cong f(x_0, y_0) + \left. \frac{\partial f}{\partial x} \right|_{x_0, y_0} (x - x_0) + \left. \frac{\partial f}{\partial y} \right|_{x_0, y_0} (y - y_0) \quad (\text{B.3})$$

As an example of the error involved in simple linearization, consider the nonlinear expression

$$z = 2x^2 + 10 \quad (\text{B.4})$$

The linear approximation to (B.4) is

$$z' = (2x_0^2 + 10) + 4x_0(x - x_0) \quad (\text{B.5})$$

Choosing an  $x_0$  of 10

$$z' = -190 + 40x \quad (\text{B.6})$$

An indication of the error introduced by the linearization is

$x$	$z$ Eq. (B.4)	$z'$ Eq. (B.6)
10	210	210
12	298	290
15	460	410

Providing we remain reasonably close to  $x_0$ , only small errors are introduced.

### Multivariable Problems

As an example of a multivariable problem, consider a typical energy balance for a heat exchanger

$$wC(T_i - T_o) + UA(T_v - T_o) = mC dT_o/dt \quad (\text{B.7})$$

where  $t$  = time;  $w$  = inlet water flow rate;  $T_i$  = inlet water temperature;  $T_o$  = outlet water temperature;  $T_v$  = steam temperature (constant); and  $C, U, A, m$  = constants. The two nonlinear terms in Eq. (B.7) are  $wT_i$  and  $wT_o$ , so we expand these about the "steady-state" values  $w_s, T_{is}, T_{os}$ , using Eq. (B.3)

$$wT_i = w_s T_{is} + w_s(T_i - T_{is}) + T_{is}(w - w_s) \quad (\text{B.8})$$

$$wT_o = w_s T_{os} + w_s(T_o - T_{os}) + T_{os}(w - w_s) \quad (\text{B.9})$$

Introducing Eqs. (B.8) and (B.9) into (B.7)

$$C[(T_{is} - T_{os})(w - w_s) + w_s(T_i - T_o)] + UA(T_v - T_o) = mC dT_o/dt \quad (\text{B.10})$$

This can be simplified by introducing the boundary condition that at the steady state,  $dT_o/dt = 0$  and  $T_i = T_{is}, T_o = T_{os}, w = w_s$ , etc. Thus Eq. (B.7) becomes

$$w_s C(T_{is} - T_{os}) + UA(T_{vs} - T_{os}) = 0 \quad (\text{B.11})$$

Subtracting Eq. (B.11) from (B.10) and introducing the deviation variables

$$\begin{aligned} \tilde{T}_i &= T_i - T_{is}, & \tilde{T}_o &= T_o - T_{os} \\ \tilde{T}_v &= T_v - T_{vs}, & \tilde{W} &= w - w_s \end{aligned}$$

$$C[(T_{is} - T_{os})\tilde{W} + w_s(\tilde{T}_i - \tilde{T}_o)] + UA(\tilde{T}_v - \tilde{T}_o) = mC d\tilde{T}_o/dt \quad (\text{B.12})$$

We now have a linear differential equation which can be Laplace transformed to yield

$$as\tilde{T}_o(s) + \tilde{T}_o(s) = b\tilde{T}_i(s) + c\tilde{T}_v(s) - d\tilde{W}(s) \quad (\text{B.13})$$

where  $a, b, c$ , and  $d$  are constants. Equation (B.13) readily lends itself to SFG construction.

To generalize, consider the vector set of first-order differential equations

$$\dot{x}_i = f_i(\mathbf{X}, \mathbf{Y}, t), \quad i = 1, 2, \dots, N \quad (\text{B.14})$$

In control parlance,  $\mathbf{X}$  is an  $N \times 1$  state vector,  $\mathbf{Y}$  an  $M \times 1$  control vector, and  $t$  is the independent variable.  $f_i$  and its derivatives relative to  $\mathbf{X}$  and  $\mathbf{Y}$  must be continuous functions of  $\mathbf{X}$  and  $\mathbf{M}$ .

Let  $\mathbf{Y}$ 's and  $\mathbf{X}$ 's denote the input and response about the point of linearization. Expanding (B.14) in a Taylor series about  $\mathbf{X}_s$  provides the changes



from the nominal point of linearization (control) over an interval

$$\tilde{x}_i = \sum_{j=1}^N (\partial f_i / \partial x_j) \tilde{x}_j + \sum_{k=1}^M (\partial f_i / \partial y_k) \tilde{y}_k + R_i(\mathbf{X}, \mathbf{Y}, t) \quad (\text{B.15})$$

where

$$\tilde{x}_i = x_i - x_{is}, \quad \tilde{x}_j = x_j - x_{js}, \quad \tilde{y}_k = y_k - y_{ks}$$

$R_i(\mathbf{X}, \mathbf{Y}, t)$  includes higher order derivatives and must be made small by keeping  $\tilde{x}_j$  and  $\tilde{y}_k$  small. Disregarding  $R_i$ , we obtain the linearized equations

$$\tilde{\mathbf{x}} = \mathbf{F}\tilde{\mathbf{X}} + \mathbf{G}\tilde{\mathbf{Y}}$$

where

$$\mathbf{F} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_N}{\partial x_1} & \frac{\partial f_N}{\partial x_2} & \cdots & \frac{\partial f_N}{\partial x_N} \end{pmatrix}, \quad \mathbf{G} = \begin{pmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \cdots & \frac{\partial f_1}{\partial y_M} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_N}{\partial y_1} & \frac{\partial f_N}{\partial y_2} & \cdots & \frac{\partial f_N}{\partial y_M} \end{pmatrix}$$

## APPENDIX C

### DERIVATION OF MASON'S RULE<sup>†</sup>

Mason's rule states that the transmittance  $T$  from input to output of a single input SFG is

$$T = (\sum P_k \Delta_k) / \Delta \quad (\text{C.1})$$

The summation is taken over all forward paths from input to output.  $P_k$  is the gain of the  $k$ th forward path; it is obtained by multiplying the gains of all branches constituting the forward path.  $\Delta$  is the determinant of the system of equations (the graph). It is obtained from

$$\Delta = 1 - \sum_m L_{m1} + \sum_k L_{k2} - \sum_l L_{l3} + \cdots \quad (\text{C.2})$$

where  $L_{m1}$  is the loop gain of the  $m$ th loop the summation being taken over all loops.  $L_{k2}$  is the product of the loop gains of all nontouching loops taken two at a time, and  $L_{l3}$  is the product of the loop gains of all nontouching loops taken three at a time, etc. The  $\Delta_k$  in Eq. (C.1) is obtained from  $\Delta$  by deleting terms containing loop gains corresponding to loops touched by

<sup>†</sup> Adapted with minor changes from "Linear Systems Theory" by L. A. Zadeh and C. A. Desoer. Copyright 1963 by McGraw-Hill, Inc. Used with permission of McGraw-Hill Book Company.

path  $P_k$ . The transmittance  $T$  in Eq. (C.1) can be used to obtain the node value  $x_k$  in a signal flow graph via

$$x_k = \sum_{j=1}^m (T_{j \rightarrow k})(I_j) \quad (\text{C.3})$$

where  $I$  is an input, there being  $m$  inputs.  $T_{j \rightarrow k}$  is the transmittance from input node  $j$  to output node  $k$ .

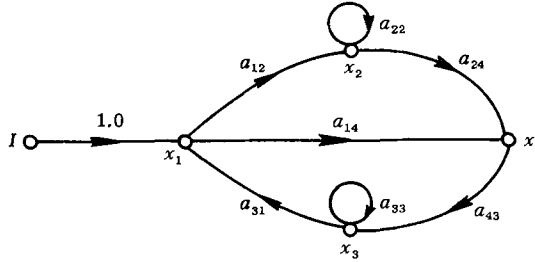


Fig. C.1. Signal flow graph.

Consider the graph of Fig. C.1. It has four loops whose respective gains are

$$\begin{aligned} L_1 &= a_{22}, & L_3 &= a_{14}a_{43}a_{31} \\ L_2 &= a_{33}, & L_4 &= a_{12}a_{24}a_{43}a_{31} \end{aligned}$$

The touching loops are  $L_1$  and  $L_4$ ,  $L_2$  and  $L_4$ ,  $L_2$  and  $L_3$ ,  $L_3$  and  $L_4$ . Consequently,

$$\begin{aligned} \Delta &= 1 - (L_1 + L_2 + L_3 + L_4) + (L_1L_2 + L_1L_3 + \underline{L_1L_4} + \underline{L_2L_3} + \underline{L_2L_4} + \underline{L_3L_4}) \\ &\quad - (\underline{L_1L_2L_3} + \underline{L_1L_2L_4} + \underline{L_1L_3L_4} + \underline{L_2L_3L_4}) - \underline{L_1L_2L_3L_4} \end{aligned}$$

The terms involving touching loops are underlined. Eliminating these,

$$\Delta = 1 - a_{22} - a_{33} - a_{14}a_{43}a_{31} - a_{12}a_{24}a_{43}a_{31} + a_{22}a_{33} + a_{22}a_{14}a_{43}a_{31} \quad (\text{C.4})$$

There are two forward paths between  $I$  and the output node  $x_4$

$$P_1 = a_{14} \quad (\text{C.5})$$

$$P_2 = a_{12}a_{24} \quad (\text{C.6})$$

Thus, the corresponding  $\Delta_1$  and  $\Delta_2$  obtained by deleting from  $\Delta$  the loops touched by  $P_1$  and  $P_2$  are

$$\Delta_1 = 1 - a_{22} - a_{33} + a_{22}a_{33} \quad (\text{C.7})$$

$$\Delta_2 = 1 - a_{33} \quad (\text{C.8})$$

Hence the transmission, with reference to Eqs. (C.1), (C.4), (C.5), (C.6), (C.7), and (C.8) is

$$T_{I \rightarrow 4} = (P_1 \Delta_1 + P_2 \Delta_2) / \Delta \quad (\text{C.9})$$

and, by Eq. (C.3), for input  $I$ , the value of node four is

$$x_4 = (T_{I \rightarrow 4})I \quad (\text{C.10})$$

To prove that Mason's rule has given us a correct result, we first recall that the determinant of a matrix  $\mathbf{A}$  is

$$|\mathbf{A}| = \sum_{n!} (\pm) a_{i_1 1} a_{i_2 2} a_{i_3 3}, \dots, a_{i_n n} \quad (\text{C.11})$$

and that, by Cramer's Rule

$$x_4 = |\mathbf{A}(4)| / |\mathbf{A}| = |\mathbf{A}(4)| / \Delta \quad (\text{C.12})$$

A definition of the terms in Eqs. (C.11) and (C.12) has been given in Appendix A.

To prove that Eqs. (C.9) and (C.12) provide a correct solution for the problem posed in Fig. C.1, let us first proceed with an evaluation of  $\Delta$ .

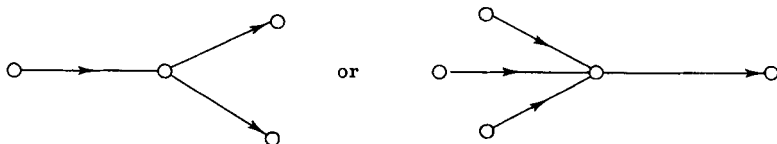
### Expansion of $\Delta$

To show the equivalence between Eqs. (C.2) and (C.11), we expand  $\Delta$  according to the definition of the determinant. Collecting terms,

$$\Delta = 1 + \sum_{\alpha} \pi_{\alpha}$$

where each  $\pi_{\alpha}$  is a product of branch gains. Let us now make some observations regarding the topology of  $P_{\beta}$ , a subgraph, having a gain product  $\pi_{\beta}$ .

1. In  $P_\beta$  not more than one branch originates from any given node or terminates at any given node. In other words, the occurrence of



is forbidden since the subgraph represents an equation such as

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$

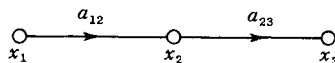
Thus  $\pi_\beta$  is a product of gains which do not contain more than one element from each row or from each column of  $A$ . Hence not more than one branch of  $P_\beta$  can terminate at any given node, or originate from any given node.

2.  $P_\beta$  is a product of loop gains of nontouching loops (having neither nodes nor branches in common). Since  $P_\beta$  is not connected it must, by (C.1), consist of nontouching loops. We must now show that each connected subgraph of  $P_\beta$  is a loop. We establish this property by contradiction. Suppose  $P_\beta$  consists of a directed path between some pair of nodes and some other nontouching configurations generated by other equations such as

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$

$$a_{31}x_1 + a_{32}x_2 + \cdots + a_{3n}x_n = b_3$$

Suppose the forward path is made of  $a_{12}$  and  $a_{23}$ , that is, originating from node 1 and terminating at node 3.



Hence no branch of  $P_\beta$  originates at 3 or terminates at 1. Hence the product cannot contain terms like  $a_{k1}$ ,  $k = 1, 2, \dots, n$  and  $a_{3k}$ ,  $k = 1, 2, \dots, n$ ; that is, no elements of column 1 or row three appear in  $P_\beta$ . The definition of a determinant implies, therefore, that  $P_\beta$  includes the 1's which are part of the elements which are in the (1, 1) and (3, 3) location in the matrix  $A$ . This implies a contradiction, because  $P_\beta$  already includes  $a_{12}$  and  $a_{23}$  elements in row 1 and column 3 and cannot include, in addition, elements at location (1, 1) and (3, 3). Therefore, if  $P_\beta$  is not connected, it consists of nontouching loops. If  $P_\beta$  is connected, it consists of a single loop.

3. The gain product in  $\pi_\beta$  is assigned a negative sign if  $\pi_\beta$  consists of an odd number of loops and a positive sign if the number of loops is even. Recall that, if in a matrix two rows and two columns are interchanged, the value of the determinant is unaffected. If we have two graphs  $\mathcal{P}$  and  $\mathcal{P}^1$  identically numbered except that the nodes of  $i$  and  $j$  are labeled  $j$  and  $i$  in  $\mathcal{P}$  and  $\mathcal{P}^1$ , respectively, the determinants are also identical. Therefore, in evaluating the determinant of a SFG, we may arbitrarily number the nodes without affecting the determinant.

Suppose  $\mathcal{P}_\beta$  consists of two nontouching loops. Let the nodes of each loop be numbered  $1, 2, \dots, l_1; l_1 + 1, \dots, l_2$ , respectively. The gain product of the loops are

$$a_{12}a_{23}a_{34} \cdots a_{l_1-1, l_1}a_{l_1, 1}a_{l_1+1, l_1+2} \cdots a_{l_2-1, l_2}a_{l_2, l_1+1}$$

This product can be assigned a sign according to the parity of the permutation

$$(l_1 + 2, l_1 + 3, \dots, l_2, l_1 + 1; 2, 3, 4, \dots, l_1, 1)$$

Since the  $l_1$ th element 1 can be brought to the position 1 by  $l_1 - 1$  interchanges of adjacent symbols, and since the  $(l_1 + l_2)$ nd element,  $l_1 + 1$ , can be brought to the  $(l_1 + 1)$ st position by  $l_2 - 1$  interchanges of consecutive symbols, the sign of the permutations ( $\pm$ ) is

$$(\pm) = (-1)^{l_1-1}(-1)^{l_2-1}$$

If each of the elements of the system matrix  $a_{ki}$  has a minus sign, the sign of the product  $\pi_\beta$  is

$$(-1)^{l_1}(-1)^{l_2}(\pm) = (-1)^2$$

If  $\mathcal{P}_\beta$  had consisted of  $\lambda$  loops, the reasoning would have led to  $(-1)$ . Therefore, any term of the expansion of  $\Delta$  is either 1 or a product  $\pi_\alpha$  of loop gains of nontouching loops times  $(-1)^{\lambda_\alpha}$ , where  $\lambda_\alpha$  is the number of loops corresponding to the product  $\pi_\alpha$ .

Conversely, any product of loop gains of nontouching loops of the graph  $\mathcal{P}$  will appear, with appropriate sign, as a term of the expansion of  $\Delta$ .

Thus Eqs. (C.4) and (C.11) are justified. It remains now to show that the numerator of Eq. (12) is justifiable.

**Evaluation of  $|A(k)|$** 

The graph of  $A(k)$ , which we will denote  $\mathcal{P}^{11}$  can be obtained from  $\mathcal{P}$  by (a) deleting all the branches that originate from the output node  $k$ , and (b) inserting a branch gain  $b$ , originating at  $k$  and terminating at 1. For purposes of this discussion we shall assume the graph and equation arrangement of Zadeh and Desoer, and write

$$\begin{array}{ccccccc} (1 - a_{11})x_1 & - & a_{12}x_2 & - & \cdots & - & a_{1n}x_n & = & b_1 \\ -a_{21}x_1 & + & (1 - a_{22})x_2 & - & \cdots & - & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ -a_{n1}x_1 & - & a_{n2}x_2 & - & \cdots & + & (1 - a_{nn})x_n & = & b_n \end{array}$$

Consider a particular product  $\pi^{11}$ . Let the forward path of  $\pi^{11}$  contain  $\lambda - 1$  branches and let its nodes be numbered  $1, 2, \dots, \lambda$ , where 1 corresponds to the input node and  $\lambda$  to the output node. Such a renumbering does not affect the value of  $A(k)$ . The gain of the forward path is  $a_{21}a_{32}a_{43}, \dots, a_{\lambda, \lambda-1}$ . We now write the matrix  $A(k)$  after the reordering, putting the branch gains of the forward path

$$\begin{array}{cccccccc} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & b_1 \\ -a_{21} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & -a_{32} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & -a_{43} & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & a_{\lambda, \lambda-1} & \cdot & \cdot \end{array} \quad \begin{array}{c} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \end{array} \quad \begin{array}{c} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \end{array}$$

$A(\lambda)$

From the form of the matrix, it is clear that the loop gain products appearing in  $\pi^{11}$  must come from the expansion of  $|A(\lambda)|$ . Consequently, if we consider in the expansion of  $|A(k)|$ , all of the terms having the product  $b_1 a_{21} a_{32} \cdots a_{\lambda, \lambda-1}$  as a common factor, we get

$$(-1)^{\lambda-1} b_1 (-a_{21}) (-a_{32}) (-a_{43}) \cdots (-a_{\lambda, \lambda-1}) |A(\lambda)|$$

where  $(-1)^{\lambda-1}$  comes from the fact that  $b_1$  belongs to row 1 and column  $\lambda$

of  $|\mathbf{A}(i)|$ . This product is thus

$$b_1 a_{21} a_{32} \cdots a_{\lambda, \lambda-1} |\mathbf{A}(\lambda)|$$

From a topological point of view,  $|\mathbf{A}(\lambda)|$  is the determinant associated with the SFG  $\mathcal{P}_\lambda$  obtained from  $\mathcal{P}$  by deleting all the nodes of the forward path and all the branches connected to this forward path. Therefore, any product  $\pi^{11}$  of the expansion  $|\mathbf{A}(k)|$  is a term of the form  $P_k \Delta_k$ . Conversely, it is easy to establish that any expression of the form  $P_k \Delta_k$  includes all of the terms of the expansion  $|\mathbf{A}(k)|$  with the correct sign. Therefore,

$$|\mathbf{A}(k)| = \sum_k P_k \Delta_k$$

and Eqs. (C.3) and (C.12) are identical, since the  $I_j$  in (C.12) is included as a node in  $P_k$ .



This page intentionally left blank

## APPENDIX D

# BOOLEAN AND MODULO-2 ALGEBRA

### Boolean Algebra

This discussion will make reference to the directed graph and the corresponding adjacency matrix of Fig. D.1.

The element  $a_{ij}$  is 1 if there is flow from  $i$  to  $j$ , otherwise it is zero. The matrix, therefore, is a listing of all one step paths between vertices, and the element  $a_{ij}$  is, in reality, a truth value where a value of 1 says yes (there is a path), and the value zero means no (there is no path). When the elements  $a_{ij}$  are understood in this context, then the product and sum of elements can be understood in terms of yes and no propositions; i.e. Boolean algebra. We now state the rules of Boolean algebra.

### MULTIPLICATION

The logical product is one if all propositions are true and zero if any proposition is false

$$a_{ij} \cdot a_{kl} \cdot \dots \cdot a_{mn} = \min(a_{ij}, a_{kl}, \dots, a_{mn})$$

Thus

$$0 \cdot 1 = 0, \quad 1 \cdot 1 = 1$$

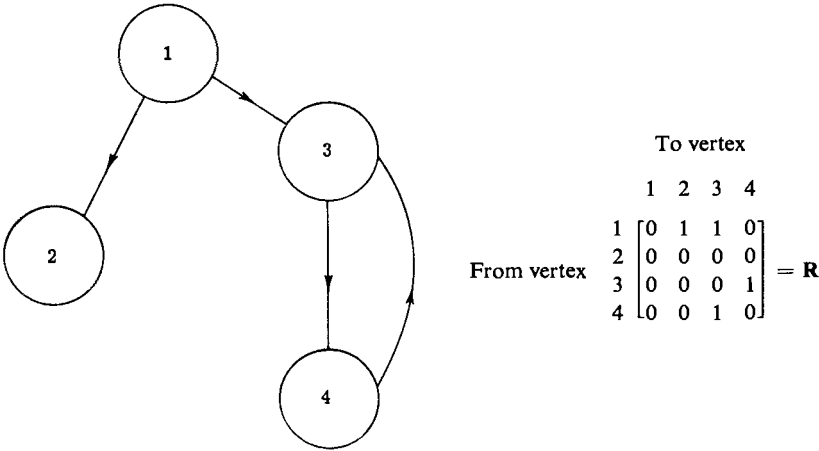


Fig. D.1. A digraph and its corresponding adjacency matrix.

ADDITION (Boolean Union)

The truth value of a sum is zero if every proposition is false, and one if any addend is 1.

$$a_{ij} \cup a_{kl} \cup a_{mn} \cup \dots = \max(a_{ij}a_{kl}a_{mn})$$

or,

$$a_{ij} + a_{kl} + a_{mn} + \dots = \max(a_{ij}a_{kl}a_{mn})$$

thus

$$0 + 1 = 1, \quad 1 + 1 = 1, \quad 1 + 0 = 1$$

Boolean addition is distributive, and the rules of ordinary algebra are obeyed with the exception that there is no meaningful analog of subtraction. Matrix multiplications are performed in the normal manner, as is vector addition. Thus, multiplication of the matrix in Fig. D.I with itself gives the second power of the adjacency matrix:

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = R^2$$

The ones in  $a_{14}$ ,  $a_{33}$ , and  $a_{44}$  indicate that there are two-step paths between the nodes 1-4, 3-3, and 4-4.

**The Field Modulo 2**

Whereas Boolean algebra finds its greatest use in the analysis of digraphs, modulo-2 algebra finds its primary use in the study of nonoriented graphs. The algebra uses the elements 0 and 1 with multiplication and addition defined as

$$0 + 0 = 0$$

$$0 + 1 = 1 + 0 = 1$$

$$1 + 1 = 0$$

$$0 \cdot 0 = 0$$

$$1 \cdot 0 = 0 \cdot 1 = 0$$

$$1 \cdot 1 = 1$$

There is no subtraction since minus signs come from the inverse elements of addition. The algebraic rules of commutation, association, and distribution are adhered to. As an example of an operation, consider the calculation of the determinant of the matrix below

$$\begin{aligned} \Delta \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} &= 1 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + 1 \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} + 1 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ &= 1(1 \cdot 0 + 0 \cdot 0) + 1(0 \cdot 0 + 0 \cdot 1) + 1(0 \cdot 0 + 1 \cdot 1) \\ &= 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 1 = 1 \end{aligned}$$

This page intentionally left blank

## APPENDIX E

### LINEAR PROGRAMMING

Linear programming is a procedure whereby one maximizes (or minimizes) a variable  $P$  which is a linear function of the variables  $x_i$ ,  $i = 1, 2, \dots, n$ .

$$P = c_1x_1 + c_2x_2 + \dots + c_nx_n \quad (\text{E.1})$$

The variables are subject to the linear inequalities (constraints)

$$\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n & \leq & b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n & \leq & b_2 \\ \vdots & & \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n & \leq & b_m \end{array} \quad (\text{E.2})$$

and

$$x_i \geq 0, \quad i = 1, 2, \dots, n \quad (\text{E.3})$$

The number of constraints  $m$  can be  $n \geq m$  or  $m \geq n$ . The maximization and minimization problems are termed "duals" and have equivalent formulations.

The inequalities in Eq. (E.2) can be converted to equalities by introducing  $m$  "slack variables." Letting  $N = n + m$ , there are now an additional  $x_m$  variables; the LP problem is

$$P = c_1x_1 + c_2x_2 + \cdots + c_Nx_N \quad (\text{E.4})$$

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1N}x_N &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2N}x_N &= b_2 \\ \vdots & \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mN}x_N &= b_m \end{aligned} \quad (\text{E.5})$$

Equations (E.3), (E.4), and (E.5) can be conveniently written in matrix form as

$$\max(\mathbf{C}^T\mathbf{X}) \quad (\text{E.6})$$

$$\mathbf{A}\mathbf{X} \leq \mathbf{B} \quad (\text{E.7})$$

$$\mathbf{X} \geq 0 \quad (\text{E.8})$$

It should be noted that many of the parameters are zero or 1.

$$\begin{aligned} c_{n+1} &= c_{n+2} = \cdots = c_N = 0 \\ a_{1,n+1} &= a_{2,n+2} = \cdots = a_{m,N} = 1 \\ a_{1,n+2} &= a_{1,n+3} = \cdots = a_{1N} = 0 \\ a_{2,n+1} &= a_{2,n+3} = \cdots = a_{2N} = 0 \\ &\vdots \\ a_{m,n+1} &= a_{m,n+2} = a_{m,N-1} = 0 \end{aligned} \quad (\text{E.9})$$

Before discussing ways in which Eqs. (E.6) and (E.7) are solved, we introduce the notion, and some theorems of convex sets.

### Convex Sets

Consider the problem of maximizing  $P$ , the "objective function" being,

$$P = 2x_1 + x_2 \quad (\text{E.10})$$

subject to the constraints

$$3x_1 + x_2 \leq 12 \quad (\text{E.11})$$

$$x_1 + 2x_2 \leq 10 \quad (\text{E.12})$$

where

$$x_1 \geq 0 \quad \text{and} \quad x_2 \geq 0 \quad (\text{E.13})$$

The geometric significance of the problem can be grasped by plotting the inequalities of Eq. (E.11) and (E.12) as equalities and noting that all possible values of  $x_1$  and  $x_2$  must lie below the shaded area  $ABC$ .

The dashed lines in Fig. E.1 are the objective functions for values of  $P = 2, 4, 6, 8$ , and  $9.2$ . The point  $B$  represents the "optimal feasible" solution; here  $x_2 = 3.6$ ,  $x_1 = 2.8$ , and  $P = 9.2$ . All higher values of  $P$  involve  $x_1$  and  $x_2$ 's which lie outside the range of feasible  $x$ 's. The points  $A$  and  $B$  also represent feasible solutions and could represent a maxima if the function  $P$  were changed, and the same constraints were maintained.

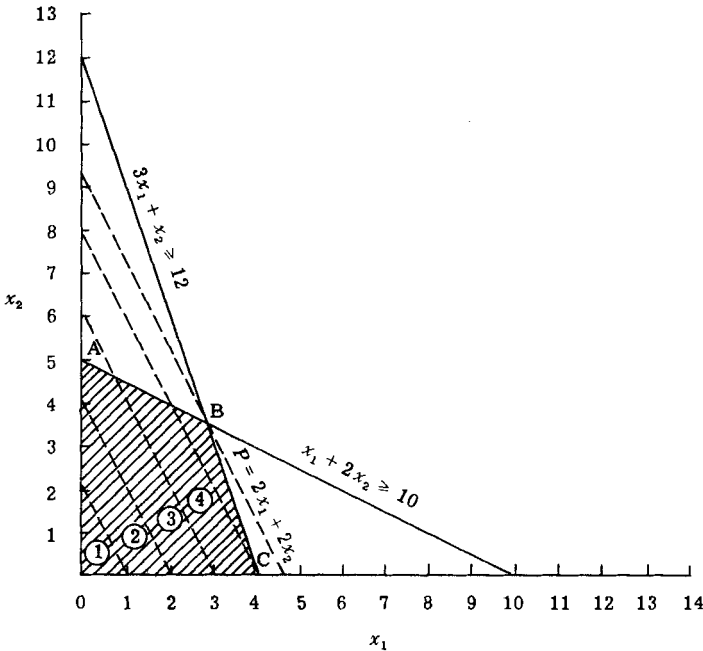


Fig. E.1. Two-variable linear optimization.

There are several points of interest regarding Fig. E.1.

1. There is a unique solution to the problem.
2. Except in some unusual cases of academic interest only, the optimum point is on one of the corners ( $A$ ,  $B$ ,  $C$ ) of the convex polygon formed by the constraint equations.



3. The points  $A(0, 5)$ ,  $B(2.8, 3.6)$ , and  $C(4, 0)$  are termed “basic feasible solutions, and  $B$  is the “optimal solution.”

4. Both the constraint and objective functions must be linear and all variables must be positive for condition 2 to hold.

5. If the inequality signs are reserved, the point  $B$  will also be the solution to the “dual” linear programming problem; find the minimum of  $P$ .

$$P = 12y_1 + 10y_2, \quad \min(\mathbf{B}^T \mathbf{Y}) \quad (\text{E.14})$$

$$3y_1 + y_2 \geq 2, \quad (\mathbf{A}^T \mathbf{Y} \geq \mathbf{C}) \quad (\text{E.15})$$

$$y_1 + 2y_2 \geq 1 \quad (\text{E.16})$$

with  $y_1 \geq 0$  and  $y_2 \geq 0$ .

That the optimum must lie on one of the corners of the convex polygon formed by the constraints will not be proven here. We will, however, summarize some of the concepts necessary for an understanding of linear programming.

Given a set of points  $S$ , among which are the vectors  $\mathbf{X}_1$  and  $\mathbf{X}_2$ , and the fact that any line connecting  $\mathbf{X}_1$  and  $\mathbf{X}_2$  is also in the set, then the set is said to be convex, and any point within the set can be related to  $\mathbf{X}_1$  and  $\mathbf{X}_2$  by

$$\mathbf{X} = \alpha \mathbf{X}_1 + (1 - \alpha) \mathbf{X}_2, \quad 0 \leq \alpha \leq 1 \quad (\text{E.17})$$

More generally, any point in a convex set can be obtained from a combination of points in the set

$$\mathbf{X} = \alpha_1 \mathbf{X}_1 + \alpha_2 \mathbf{X}_2 + \cdots + \alpha_i \mathbf{X}_i, \quad \alpha_i > 0, \quad \sum \alpha_i = 1 \quad (\text{E.18})$$

By this definition Fig. E.2a is a convex set, but Fig. E.2b is not. (A line connecting points 2 and 4, for example, would fall outside polygon  $B$ .)

An extreme point of a convex set is one which does not lie on a line between any other two points, i.e., points 1,  $\dots$ , 6 in Fig. E.2a. From the theory of convex sets, it can be shown that the optimal feasible solution

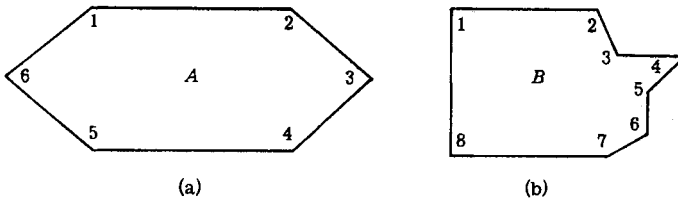


Fig. E.2. (a) Convex and (b) nonconvex sets.

must lie at one of the extreme points of the convex set formed by the constraint equation, and that it cannot lie in the interior of the polygon. Thus the system of equations need be solved only at the extreme points, and one of these will yield the optimal feasible solution. Proofs of this and the fundamental theorems of linear programming central to the method of solution known as the *simplex method* of Danzig may be found in most books on linear programming.

### Simplex Method

In general, the simplex method is an iterative process whereby we begin the calculations with a set of values for the independent variables that satisfy the constraints, and in which each iterative cycle seeks another set of values that improve the desired objective function. It is essentially a structured trial-and-error solution which is so formulated that a systematic mathematical treatment is followed. The procedure is to move repeatedly from an extreme point (basic feasible solution) along an "edge" to an adjacent point with a larger value of  $P$ .

One usually begins by setting the  $x_n$  variables to zero and solving Eq. (E.5) for the  $m$  slack variables ("the basis"). This produces a value of 0 for  $P$ . We then systematically replace the  $x_m^*$ 's in "the basis" always selecting  $x$ 's which increase  $P$ . The algebraic manipulations will now be demonstrated by study of an LP problem posed by Hillier and Lieberman [1]

$$\begin{array}{ll}
 P = 3x_1 + 5x_2 & P = 3x_1 + 5x_2 \\
 x_1 \leq 4 & x_1 + x_3 = 4 \\
 x_2 \leq 6 & x_2 + x_4 = 6 \quad (\text{E.19}) \\
 3x_1 + 2x_2 \leq 18 & 3x_1 + 2x_2 + x_5 = 18 \\
 x_1 \geq 0, \quad x_2 \geq 0 & x_m \geq 0, \quad m = 1, \dots, 5
 \end{array}$$

The steps in the solution are

*Step 1.* Introduce the slack variables. These are shown to the right in Eq. (E.19) as  $x_3, x_4, x_5$ .

*Step 2.* Select the slack variables to be the initial basic variables and solve the constraint equations.

This is the initial basic feasible solution ( $x_1 = 0, x_2 = 0, x_3 = 4, x_4 = 6, x_5 = 18$ ).

*Step 3.* Select as a new entering basic, the nonbasic variable which increases  $P$  the fastest.

In this case  $x_2$  is chosen.

*Step 4.* Select a variable to leave the basis.

Since  $x_3$ ,  $x_4$ , and  $x_5$  are all candidates and

$$\begin{aligned}x_3 &= 4 - x_1 \\x_4 &= 6 - x_2 \\x_5 &= 18 - 3x_1 - 2x_2\end{aligned}$$

we choose the one which first turns infeasible (negative) as  $x_2$  increases. Since  $x_1 = 0$ , this is clearly  $x_4$ .

*Step 5.* Determine the new basic feasible solution: Solve for the basic variables in terms of the nonbasic by the Gauss-Jordan method.

The original equations with the slack variables as a basis can be arranged in a Gauss-Jordan type of array as

$$\left. \begin{array}{rcll} (0) & P - 3x_1 - 5x_2 & = & 0 \\ (1) & x_1 & + x_3 & = 4 \\ (2) & x_2 & + x_4 & = 6 \\ (3) & 3x_1 + 2x_2 & + x_5 = 18 \end{array} \right\} \approx \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \\ 6 \\ 18 \end{bmatrix}$$

Since the nonbasic variables  $x_1$  and  $x_2$  are zero, we obtain the G-J matrix to the right.

In accordance with step 4, we now remove  $x_4$  from the basis and replace it with  $x_2$ . Interchanging columns 3 and 5, and doing a G-J reduction produces (with  $x_1$  and  $x_4$  equal to zero)

$$\left. \begin{array}{rcll} (0) & P - 3x_1 + 5x_4 & = & 30 \\ (1) & x_1 & + x_3 & = 4 \\ (2) & x_4 & + x_2 & = 6 \\ (3) & 3x_1 - 2x_4 & + x_5 = 6 \end{array} \right\} \approx \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P \\ x_3 \\ x_2 \\ x_5 \end{bmatrix} = \begin{bmatrix} 30 \\ 4 \\ 6 \\ 6 \end{bmatrix}$$

The values of the basic variables are  $x_3 = 4$ ,  $x_2 = 6$ ,  $x_5 = 6$ , and  $P = 3x_1 + 5x_2 = 30$ .

*Step 6.* Repeat steps 4 and 5 until  $P$  cannot be increased.

The objective function from step 5,  $P = 30 + 3x_1 - 5x_4$  can be increased by insertion of  $x_1$  into the basis. The variable to be replaced is  $x_5$  since this must be less than 2, whereas the upper limit to  $x_3$  is 4.

Thus, substituting  $x_1$  for  $x_5$  into the basis produces the array

$$\left. \begin{array}{rcl} P + 3x_4 & x_5 & = 36 \\ \frac{2}{3}x_4 - \frac{1}{3}x_5 + x_3 & & = 2 \\ x_4 & + x_2 & = 6 \\ -\frac{2}{3}x_4 + \frac{1}{3}x_5 & + x_1 & = 2 \end{array} \right\} \approx \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P \\ x_3 \\ x_2 \\ x_1 \end{bmatrix} = \begin{bmatrix} 36 \\ 2 \\ 6 \\ 2 \end{bmatrix}$$

The basic feasible solution now is  $x_1 = 2$ ,  $x_2 = 6$ ,  $x_3 = 2$ ,  $x_4 = 0$ ,  $x_5 = 0$ . The corresponding value of  $P$  cannot be increased by rotating  $x_4$  or  $x_5$  into a new basis since  $P = 36 - 3x_4 - x_5$ . Thus we have found the optimal feasible solution.

The foregoing manipulations can be neatly written in terms of simplex tableaux, which are readily transformed into computer algorithms.

#### REFERENCE

1. F. S. Hillier and G. J. Lieberman, "Introduction to Operations Research," p. 144. Holden-Day, San Francisco, California, 1967.

This page intentionally left blank

## THE COATES FLOW GRAPH

277

To construct a Coates graph  $G_c$ , the system of Eqs. (F.1) is written as

$$\begin{aligned}
 a_{11}x_1 + a_{12}x_2 + \cdots - b_1 &= 0 \\
 a_{21}x_1 + a_{22}x_2 + \cdots - b_2 &= 0 \\
 \vdots & \\
 a_{n1}x_1 + a_{n2}x_2 + \cdots - b_n &= 0
 \end{aligned} \tag{F.3}$$

For the case of  $n = 2$ , the graph becomes as shown in Fig. F.1. This graph has the following features: (a) Each node except the source represents an equation; (b) The  $k$ th equation in the system is obtained by equating to zero the sum of the products of the branch gains of the incoming branches to  $x_k$  times the respective nodes from which the branches originate.

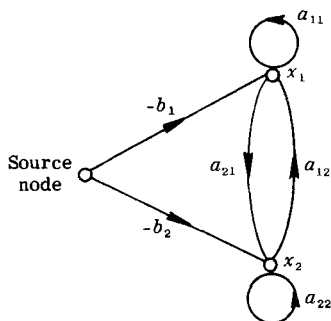


Fig. F.1. The Coates graph of Eq. (F.1).

Before Coate's gain formula can be understood, four new definitions must be introduced

1. The *subgraph*  $G_{co}$  of a flow graph  $G_c$  is formed by deleting the source node and its outgoing branches.
2. A *connection* of the subgraph  $G_{co}$  is a graph such that (a) each node of  $G_{co}$  is included; (b) each node has only one branch entering and one branch leaving.
3. The *connection gain*  $P_{co}$  of a connection is the product of all branch gains of that connection.
4. A *one-connection* is a subgraph of  $G_c$  containing all its nodes, subject to the conditions that only single (forward) connections between nodes are enumerated. Self-loops may also be listed, but no two nodes may have more than a single- (in and out) path connection.

The Coate's gain formula states that

$$x_i = \frac{\sum_k (-1)^{L_k} (P_{1 \rightarrow i})_k}{\sum_l (-1)^{L_l} (P_{co})_l} \quad (\text{F.4})$$

where  $(P_{1 \rightarrow i})_k$  = the one-connection gain from the source node to  $x_i$  summed over the  $k$  one-connections;  $L_k$  = loops in the  $k$ th one-connection;  $(P_{co})_l$  = the connection gain of the  $l$ th connection summed over all  $l$  connections;  $L_l$  = loops in the  $l$ th connection.

### An Example

Equation (F.4) will now be used to solve an example problem from Chan [3, p. 245]. For the set of equations

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{22}x_2 + a_{23}x_3 &= 0 \\ a_{31}x_1 + a_{33}x_3 &= b_3 \end{aligned} \quad (\text{F.5})$$

the Coate's graph, the  $G_{co}$ , the set of all connections of  $G_{co}$ , and the set of all one-connections of  $G_c$  are given in Fig. F.2.

With  $i = 2$ , we have for  $x_2$

$$\begin{aligned} (P_{1 \rightarrow 2})_{k=1} &= (-b_1 a_{31} a_{23}), & L_{k=1} &= (-1)^0 \\ (P_{1 \rightarrow 2})_{k=2} &= (-b_3 a_{23} a_{11}), & L_{k=2} &= (-1)^1 \\ (P_{co})_{l=1} &= a_{11} a_{22} a_{33}, & L_{l=1} &= (-1)^3 \\ (P_{co})_{l=2} &= a_{31} a_{13} a_{22}, & L_{l=2} &= (-1)^2 \\ (P_{co})_{l=3} &= a_{12} a_{31} a_{23}, & L_{l=3} &= (-1)^{-1} \end{aligned}$$

Thus

$$x_2 = \frac{(-1)^0(-b_1 a_{31} a_{23}) + (-1)^3(a_{11} a_{22} a_{33})}{(-1)^3(a_{11} a_{22} a_{33}) + (-1)^2(a_{31} a_{13} a_{22}) + (-1)^1(a_{12} a_{31} a_{23})}$$

A Mason graph may be transformed into a Coate's graph and vice versa by rules set forth by Chan [3, p. 258]. This text also contains a complete discussion of the Chan Bapna-modified Coate's graph and the Chan-Mai graph.



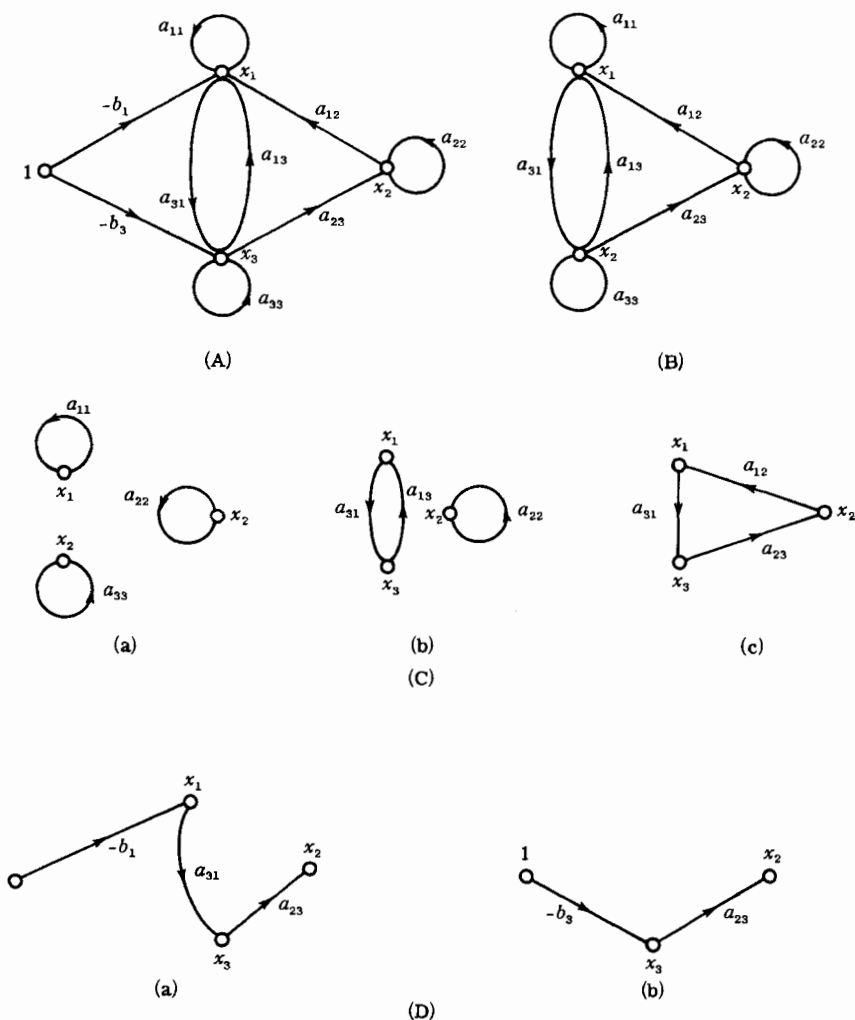


Fig. F.2. (A) Coates' graph  $G_e$ . (B)  $G_{e0}$  of the graph  $G_e$ . (C) The three sets of connections of  $G_{e0}$ . (D) The two sets of one-connections of  $G_e$ .

## REFERENCES

1. C. L. Coates, Flow-graph solution of linear algebraic equations. *IRE Trans. Circuit Theory* 6, 170-187 (1959).
2. C. A. Desoer, The optimum formula for the gain of a flow graph, or a simple derivation of Coates' formula. *Proc. IRE* 48, 883-889 (1960).
3. S. Park Chan, "Introductory Topological Analysis of Electrical Networks," Chapter 7. Holt, New York, 1969.

## APPENDIX G

### THE FIBONACCI SEARCH METHOD

The Fibonacci search technique is a highly effective technique for finding the minimum or maximum of a unimodal function in a given interval. To apply the method, the user first decides how many points of the function under study he wants to sample and the total interval over which the optimum is to be sought. A table of Fibonacci numbers is then used to determine the number of interval divisions, and the points at which the function is to be sampled.

Consider the problem of finding the optimum of  $f = x_1x_2 = f(\lambda)$ <sup>†</sup>

$$x_1 = \frac{10}{3} + \lambda, \quad x_2 = \frac{10}{3} - \frac{1}{2}\lambda$$

We wish to maximize  $f$  with respect to  $\lambda$  in the interval  $[0, 11/3]$  and we arbitrarily choose to evaluate  $f(\lambda)$  at six locations, the highest value of  $f(\lambda)$  at these six locations being taken as the maximum.

To determine the number of divisions in the interval, we enter Table G.1 at the row marked 6, and read 21 in column two. Thus  $\Delta\lambda = (11/3)/21 = 11/63$ .

<sup>†</sup> This problem was previously considered in Chapter 8.

TABLE G.1  
FIBONACCI SEARCH<sup>a</sup>

Number of trials	Number of divisions	Location of first trial
2	3	1
3	5	2
4	8	3
5	13	5
6	21	8
7	34	13
8	55	21
9	89	34
10	144	55
11	233	89
12	377	144
13	610	233
14	987	377
15	1,597	610
16	2,584	987
17	4,181	1,597
18	6,765	2,584
19	10,946	4,181
20	17,711	6,765
21	28,657	10,946
22	46,368	17,711
23	75,025	28,657
24	121,393	46,368

<sup>a</sup> D. J. Wilde, *Ind. Eng. Chem.* **57**, 23 (1965).

Consulting Table G.1 again, we see that our first functional evaluation should be carried out at the eight interval.

*First trial*

$$\lambda_1 = (8)(\Delta\lambda) = 88/63$$

$$f_1 = \left(\frac{10}{3} + 88/63\right)(10/3 - \frac{1}{2}(99/63)) = 12.46$$

These values are entered in Table G.2.

TABLE G.2

Intervals	Trials	$\lambda$	$f$
1			
2			
3			
4			
5	3	55/63	12.2
6			
7			
8	1	88/63	12.46
9	6	99/63	12.50
10	4	110/63	12.50
11	5	121/63	12.46
12			
13	2	143/63	12.3
14			
15			
16			
17			
18			
19			
20			
21			

*Second trial.* In the Fibonacci search method, the second trial is located symmetrically with respect to the first eight divisions below the maximum

$$\lambda_2 = (\Delta\lambda)(21 - 8) = 143/63$$

$$f_2 = 12.3$$

*Third trial.* The assumption that the dependent variable is unimodal permits the elimination of the interval (13–21). The third trial is placed symmetrically with respect to the first 8 places below interval 13, at location 5.

$$\lambda_3 = (\Delta\lambda)(13 - 8) = 55/63, \quad f_3 = 12.2$$

*Fourth trial.* We eliminate from consideration the interval (1–5) the maximum exists in the interval (5–13) and  $\lambda$  is between  $\lambda = 55/63$  and  $\lambda = 143/63$ . Consulting Table G.1

$$\lambda_4 = (\Delta\lambda)(13 - 3) = 110/63, \quad f_4 = 12.5$$

*Fifth trial.* The maximum is now in the interval  $(88/63-143/63)$  or  $(8-13)$ .

$$\lambda_5 = (\Delta\lambda)(13 - 2) = 121/63, \quad f_5 = 12.46$$

*Sixth trial.* Since  $f_4 > f_5$ , the maximum exists in the interval  $(88/63, 121/63)$  or  $(8, 11)$ .

$$\lambda_6 = (\Delta\lambda)(11 - 2) = 99/63, \quad f_6 = 12.5$$

This confirms the value obtained during the second trial using the method of feasible directions (Chapter 8).

## APPENDIX H

# GLOSSARY OF GRAPH NOMENCLATURE

A listing of graph-related terms in this book along with their definitions is presented in this appendix. Where possible, equivalent nomenclature is given.

*Across variables*: Potential variables (velocity, voltage, pressure).

*Acyclic graph (cascade)*: A directed graph having no cycles (loops).

*Adjacency matrix*: See Table 10.1, p. 176.

*AND/OR tree*: Generalization of a search tree.

*Arc*: See Branch.

*Associated matrix*: See Table 10.1, p. 176.

*Betti number*: See Circuit rank.

*Bipartite graph*: A graph which can be partitioned such that every edge has an end point in each set.

*Boolean matrix*: See Table 10.1, p. 176.

*Branch (edge, arcs, links, cord)*: A line connecting two vertices of a graph and containing no other vertex. A directed branch is a branch with an arrow.

*Branch and bound*: Technique to reduce the number of solutions enumerated.

*Branch gain (branch transmittance, transfer function, gain, transmittance function)*: An operator  $a_{ij}$  which maps node  $x_i$  onto  $x_j$ , i.e.,  $x_i = a_{ij}x_j$ . (The term transfer function is usually used if  $a_{ij}$ ,  $x_i$ , and  $x_j$  are functions of  $s$ .)

*Branch transmittance*: See Branch gain.

- Cardinality*: Number of elements in a set.
- Cascade graphs*: See Acyclic graph.
- Chain progression*: See Path.
- Chord (link)*: An element of a complement of a tree.
- Circuit matrix*: See Table 10.1, p. 176.
- Circuit rank (cyclomatic number, nullity, connectivity, first Betti number)*: Minimum number of edges which must be removed to render a cyclic graph acyclic.
- Closed node*: Node of a search tree which can no longer be processed.
- Complement (cotree)*: The edges left out of a system graph when a tree is formed.
- Complementary variables*: A pair of across and through variables.
- Complete graph*: Graph containing an edge from each vertex to every other one.
- Completion*: In integer linear programming, variables not yet assigned.
- Connected graph*: One in which each pair of vertices is connected by at least one edge.
- Connection matrix*: See Table 10.1, p. 176.
- Connectivity*: See Circuit rank.
- Cotree*: See Complement.
- Current edge*: Edge under consideration in some step of an algorithm.
- Current node*: Node under consideration in some step of an algorithm.
- Cut-set*: A set of edges of a connected graph  $G$  such that the removal of these edges reduces the rank of  $G$  by one.
- Cut-set matrix*: See Table 10.1, p. 176.
- Cycle*: See Loop.
- Cyclic graphs*: Graphs containing loops.
- Cyclomatic number*: See Circuit rank.
- Degrees of freedom*: See Node degree.
- Degree of a vertex*: Number of edges incident to a vertex.
- Digraphs*: See Signal flow graph.
- Directed branch*: See Branch.
- Directed graph*: See Signal flow graph.
- Directed network*: See Signal flow graph.
- Dummy node*: A branch having a transmission function of unity, and which is inserted into a graph.
- Edge*: See Branch.
- Edge progression*: See Path.
- End vertex*: An output or input node.
- Evaluation function (scoring function)*: Function which accounts for the local situation during a search.
- Feedback*: See Loop.
- Forbidden edge*: Edge banned from all subsequent solutions in a branch of a search tree.
- Forest*: A connected graph of trees (free from circuits).
- Forward edge*: One leading to a higher numbered node.
- Forward path (direct path)*: A path through a series of nodes where successive nodes are of higher order.
- Free edge*: Edge neither forbidden nor imposed.
- Gain*: See Branch gain.
- Hamiltonian circuit*: See Tour.
- Heuristical*: Usually means nonoptimal, or not rigorous.
- Imposed edge*: Edge required in all subsequent solutions in a branch of a search tree.

*Incidence matrix*: See Table 10.1, p. 176.

*Index*: The minimum number of nodes which must be split to render a cyclic graph acyclic.

*Indicator*: Value of the evaluation function attached to a node of a search tree.

*Input node*: See Source.

*Junction point*: See Node.

*Level*: Distance from the root for a node of a search tree.

*Link*: See Branch.

*Logic tree*: A graph used for enumerating possibilities or decisions.

*Loop gain*: A product of the branch gains in a loop.

*Markov matrix*: See Table 10.1, p. 176.

*Maximal cycle*: A cycle such that every other cycle in the graph either is contained in it or has no common vertices with it.

*Network*: See Signal flow graph.

*Node degree (degrees of freedom)*: Number of edges associated with a node.

*Nodes (vertices, junction points, terminals)*: The variables or inputs of the linear equations defined by a signal flow graph.

*Nullity*: See Circuit rank.

*Open path (cascade)*: A path where no node is crossed more than once.

*Output node*: See Sink.

*Output variables*: The dependent variables in a set of equations.

*Partial tour*: Subset of a tour.

*Path (edge progression, chain progression)*: A set of at least two connected (and ordered) branches,  $a_{ij}a_{jk}a_{kl}\dots$

*Path gain*: A product of branch gains along a path.

*Pending node*: Not yet processed node of a search tree.

*Pruning*: Discarding certain nodes of a search tree according to some criterion.

*Pseudo-Boolean*: Denotes a problem with Boolean variables and integer coefficients.

*Rank*: The rank of a graph is  $(v - k)$  where  $v$  are the vertices and  $k$  the connected trees.

*Reachability matrix*: See Table 10.1, p. 176.

*Residual graph (subgraph)*: A graph formed by elimination of some nodes.

*Reverse edge*: One leading to a lower numbered node.

*Reverse path (inverse path)*: A path along which the nodes are of successively lower order.

*Root*: The starting node of a tree.

*Scoring function*: See Evaluation function.

*Self-loop (feedback path, closed path)*: A path originating and terminating on the same node.

*Signal flow graph (directed network, digraph, directed graph)*: A graph of directed branches interconnected at points called nodes. It defines uniquely a set of linear algebraic or differential equations.

*Sink (output node)*: A node with only incoming branches.

*Source (input node)*: A node having only outgoing branches.

*Structural matrix*: See Table 10.1, p. 176.

*Subtour*: cycle of length less than the length of a tour.

*Terminal*: a node which represents a system interface.

*Terminal graph*: A tree.

*Terminal variables*: Variables associated with a given terminal.

*Through variables*: Flow variables (current, force, flow).



*Tour*: Simple cycle containing all the vertices of a graph.

*Transfer function*: See Branch gain.

*Transition matrix*: See Table 10.1, p. 176.

*Transmission function*: See Branch gain.

*Transmittance*: See Branch transmittance.

*Tree (cascade)*: A connected graph having no loops.

*Undirected graph*: A graph in which the edges are lines without arrows (direction).

*Vertex matrix*: See Table 10.1, p. 176.

*Vertices*: See Nodes.

## APPENDIX I

### COMPUTER PROGRAMS FOR CHAPTERS 1-6

This appendix contains a brief description of the computing system developed to implement the techniques presented in the text (Chapters 1-6). The fully documented program is available from the author.

The system is composed of eighteen subroutines; four subprograms with multiple entry points and seven with single entry points. These subroutines and their primary functions are summarized below.

The system was developed in subroutine form so that it would be readily adaptable to a wide variety of computational schemes and problem types. This structure requires, however, that certain subroutines not be executed unless their "prerequisite" subroutines are executed first. The following list indicates the prerequisite subroutines for each subroutine since it shows the source of information used by each subroutine.

DATAIN	Reads data which defines a flow graph or signal flow graph.
DATOUT	Echo checks data read by subroutine DATAIN.
LOOPS	Finds all the loops in the flow graph read by subroutine DATAIN.
PATHS	Argument (INPUT). Finds all the paths in the flow graph which start at a specified INPUT node.

CLEAN	Removes extraneous information from the loop list structure generated by subroutine LOOPS.
DELTA	Uses the loops found by subroutine LOOPS to generate the delta list structure and evaluate the system determinant.
OGAIN	Argument (OUTPUT). Uses the paths found by subroutine PATHS and the delta list structure generated by subroutine DELTA to evaluate the transfer function relating the specified INPUT node to the specified OUTPUT node.
TRACE	Argument (I). Retrieves the nodes associated with the Ith entry in the list structure and prints them in sequence.
TRACE 1	Arguments (I, K). Prints elements PP(K) through PP(PPL) in sequence.
PRINTR	Printer control, tops page.
LOOPP	Prints the number of loops detected by subroutine LOOPS and enumerates them.
DELTAP	Argument (EXT). Prints the delta list structure and delta if EXT is nonzero, otherwise only delta (value of the system determinant) is printed.
PATHP	Argument (OUTPUT). Itemizes all paths starting at an INPUT node (defined by a call to subroutine PATHS) and ending at the OUTPUT node specified. If OUTPUT is zero, all paths starting at the INPUT node are printed.
OGAINP	Arguments (INPUT, OUTPUT). Prints the steady state gain and transfer function between nodes INPUT and OUTPUT.
ROOTS	Arguments (see subroutine ROOTS). Determines the real and complex roots of a polynomial via Barstow's method.
INVERT	Arguments (see subroutine INVERT). Multiplies a given transfer function by $1/s$ and inverts to the time domain.
BODE	Arguments (see subroutine BODE). Generates the values for a Bode diagram which correspond to a given transfer function.
SENSE	Arguments (see subroutine SENSE). Computes the network functions which define the sensitivity of a given transfer function with respect to a given parameter.

### Data Structure

Most of the subroutines do not have calling arguments—the greater part of the communication between subroutines being handled through COMMON. There are two COMMON blocks, MASON1 and MASON2, which contain both input and working quantities. In order to facilitate changes in storage allocation, all of the dimensioned variables were declared together in COMMON block MASON2. Any MAIN program or additional subroutine which needs to access quantities stored in either COMMON area must contain the appropriate COMMON declarations.

As discussed in Chapter 3 the basic media of storage is the list structure. In this system all of the necessary list structures are contained in one array,  $P$  (PLENTH, 4), which is partitioned into a number of smaller arrays of the same width, but of varying length. This is necessary in order to use the available storage efficiently since it is impossible to know beforehand how long each of the list structures will be. When handled in this fashion the problem of one array being too long while another is too short will never arise. A vector  $G$ , having PLENTH elements is used to store the numerical value associated with each entry in the various list structures. Figure 3.5 shows how the  $P$  array and the  $G$  vector are partitioned and what type of information is stored in each location.

This page intentionally left blank

## APPENDIX J

### COMPUTER PROGRAMS FOR CHAPTER 7

This appendix contains a brief description of the computing system for implementing the LP and MFD procedures. The programs are available from the author.

#### System Structure for LP

The LP program consists of a main program and six subroutines, four of them, LOOPS, CLEAN, DELTA, and OGAIN have been described in Appendix I.

The basic structure, referring to the LP solution procedure developed in Chapter 7, is shown in Fig. J.1.

The MAIN PROGRAM executes the simplex procedure.

SUBROUTINE DGENER draws the flow graph in terms of starting nodes, ending nodes and the branch gains.

SUBROUTINE MASONS serves as the main program in the original Mason's routine.

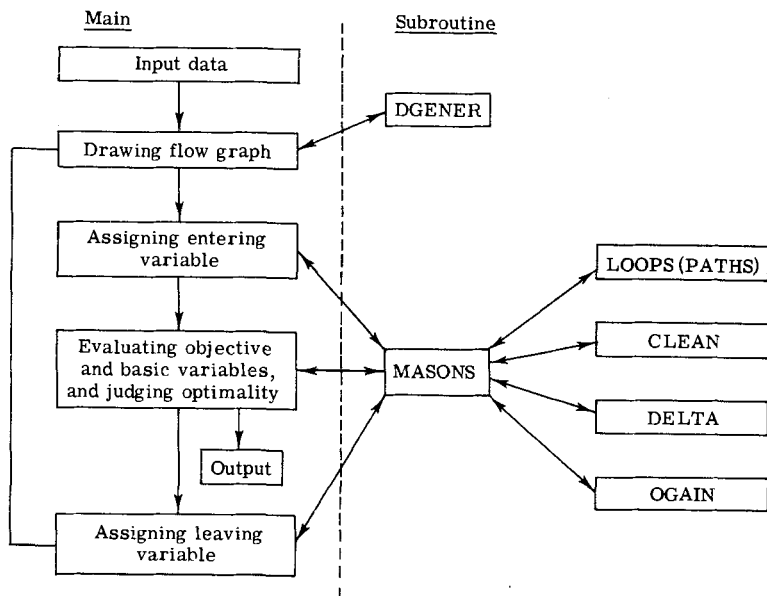


Fig. J.1

### System Structure for MFD

This program is constructed for the nine heat exchanger system described in Chapter 7. The dimensions and formats are highly specific.

The signal flow graph for the set of equations is drawn where  $I$  in  $HO(I)$  is the number of the node which represents  $HO(I)$  from Fig. 7.8. Linearizations of the heat transfer equations are shown in the section for SUBROUTINE DGAIN.

The inequality constraints of temperatures are checked in SUBROUTINE FMALFA so that a direct search by SUBROUTINE FIBONA will be done only in the feasible region; otherwise correction of the direction by LP is necessary. The coefficients and constants (right-hand side of constraints) are calculated before calling the LP subroutine in SUBROUTINE DCOEFF. The coefficients for the constraint equations in the LP do not change for each iteration of the gradient method, so these are given as input data in the MAIN PROGRAM. SFGs cannot be used to evaluate  $A$ 's and  $J$  because of nonlinearities, so they are evaluated directly in FUNCTION VALUE at each iteration of the gradient method and for the direct minimum search by SUBROUTINE FIBONA.

The main program of the LP solution has been slightly changed so that it can be used as a SUBROUTINE, MASNLP ( $X$ ), where the argument  $X$  is the solution to the LP.

## BIBLIOGRAPHY

The books in the following bibliography are wholly or mostly on graph theory.

- Bellman, R., Cooke, K. L., and Lockett, J. A., "Algorithms, Graphs, and Computers." Academic Press, New York, 1970.
- Berge C., "The Theory of Graphs and Its Applications." Methuen, London, 1962.
- Busacker, R. G. and Saaty, T. L., "Finite Graphs and Networks." McGraw-Hill, New York, 1965.
- Chan, S. Park, "Introductory Topological Analysis of Electrical Networks." Holt, New York, 1969.
- Chen, Wai-kai, "Applied Graph Theory." North-Holland, Amsterdam, 1971.
- Ford L. K. and Fulkerson, D. R., "Flows in Networks." Princeton Univ. Press, Princeton, New Jersey, 1962.
- Harary, F., "Graph Theory." Addison-Wesley, Reading, Massachusetts, 1969.
- Harary, F., Norman, R. L., and Cartwright, D., "Structural Models." Wiley, New York, 1965.
- Iri, M., "Network Flow and Transportation Scheduling." Academic Press, New York, 1969.
- Kaufman, A., "Graphs, Dynamic Programming and Finite Games." Academic Press, New York, 1967.
- Lorens, C. S., "Flowgraphs." McGraw-Hill, New York, 1964.



- Maxwell, L. M., and Reed, M. B., "The Theory of Graphs." Pergamon, Oxford, 1971.
- Ore, O., "Graphs and Their Uses." Random House, New York, 1963.
- Robichaud, L. P. A., Boisvert, M., and Robert, J., "Signal Flow Graphs and Applications." Prentice Hall, Englewood Cliffs, New Jersey, 1962.
- Seshu, S., and Reed, M. B., "Linear Graphs and Electrical Networks." Addison Wesley, Reading, Massachusetts, 1961.

# INDEX

## A

Abrahams, D. J., 21  
Absorption column, 71  
Acyclic graph, 6, 285  
Acyclic path, 161  
Across variable, 183, 285  
Active redundancy, 224, 225  
Addition rule, 10  
Adjacency matrix, 176, 179, 266, 285  
Adjoint, 249  
    matrix, 100  
AGAINP, subroutine, 88  
Alderson, G. M., 21  
Amplitude ratio, 47  
AND/OR trees, 216, 285  
Analog curves, 97  
Anomalous rankings, 158  
Arc, 285  
Arithmetic mean temperature, 138, 147  
Ashour, S., 221  
Assignment problem, 167  
Associated matrix, 176, 285

## B

Balas, E., 221, 222  
Balinski, M. L., 221  
Bandwidth, 50

Barthes, Jean-Paul, 189  
Base point, 125  
Basic variable, 110  
Batts, J. R., 244  
Beadles, R. L., 244  
Bellman, R., 163, 174, 285, 295  
Bellmore, M., 221  
Benayoun, R., 208, 221  
Benning, C. J., 244  
Berge, C., 192, 219, 221, 295  
Berztiss, A. T., 25, 43  
Betti number, 170, 285  
Bidding contest, 209  
Bilinear form, 57  
Bipartite graphs, 171, 285  
Block diagrams, 2  
Bode diagram, 48  
Bode stability criteria, 49  
Bode, subroutine, 90, 290  
Boisvert, M., 12, 13, 295  
Bollinger, R. E., 52  
Boolean algebra, 175, 265  
Boolean matrix, 176, 285  
Boolean union, 266  
Bound and scan, 189  
Bradley, C. H., 222  
Branch, 285  
    and bound, 167, 189

Branch (*continued*)

- and exclude, 189
- gain, 2, 285
- and prune, 189
- search, 189
- transmittance, 285

Brown, D. B., 244

Busacker, R. G., 295

Bypass, 60

## C

Cardinality, 286

Cartwright, D., 174, 295

Cascade, 6, 286

Chain progression, 161, 286

Chan-Bapna graph, 279

Chan-Mar graph, 279

Chan, S. Park, 69, 280, 295

Characteristic determinant, 100

Characteristic equation, 51

Charlton, J. M., 222

Chen, Wai-kai, 295

Chord, 162, 286

Christensen, J. H., 108, 222

Circuit, 6, 161

matrix, 176, 184, 286

rank, 170, 286

CLEAN, subroutine, 43, 81, 88, 290, 293

Closed chain, 161

Closed loop, 49

Closed node, 191, 286

Closed path, 6

Coates, C. L., 28

Coates graph, 277

Cofactors, 249

Cold standby module, 225

COMIT, subroutine, 10, 21

Compatible offer, 212

Complement, 286

Complementary variable, 183, 286

Complete graph, 286

Computer programs, 290, 293

Concave functions, 147

Connected graph, 286

Connection, 278

gain, 278

matrix, 14, 176, 286

Connectivity, 170, 286

Constraints, linear, 269

Continuous tank reactor, 83

Contour tangent, 124

Converter, 239

Convex functions, 147

Convex polygon, 271

Convex sets, 270

Cooke, K. L., 174, 295

Cotrees, 162, 286

Coughanowr, D. R., 108

Coulter, K. E., 244

Cramer's rule, 250, 259

Critical values, 51

Current edge, 286

Current node, 194, 286

Cut-set, 171, 286

matrix, 176, 185, 286

Cycle, 286

Cyclic graph, 6, 286

Cyclomatic number, 170, 286

## D

DATAIN, subroutine, 42, 74, 81, 88, 102, 289, 293

DATOUT, subroutine, 289

Davidson, E. S., 222

DCOEFF, subroutine, 294

DDC safety system, 238

Death, C. C., 222

Degree of a vertex, 286

Degrees of freedom, 286

Delayed signal, 85

DELTA, subroutine, 42, 74, 81, 101

DELTAP, subroutine, 102, 290

DEND, row, 43

DEND 1, row, 43

Design variable, 172

Desoer, C. A., 263, 277

DGAIN, subroutine, 294

DGENER, subroutine, 293

Diagonal matrix, 245

Direct path, 6

Directed branch, 286

Directed graphs, 1, 286

Directed network, 286

Direction vector, 121

Disconnecting sets, 171

Distillation column, 50  
 Doig, A., 206, 221  
 Driving function, 2  
 DSTART, row, 43  
 Duals, 269  
 Dual simplex, 120  
 Dummy node, 286

## E

Eastman, W. L., 203, 221  
 Edge, 286  
   current, 286  
   forbidden, 194, 286  
   forward, 29, 286  
   free, 194, 286  
   imposed, 174, 286  
   progression, 161  
   vertex, 286  
 Edwards, D. J., 21  
 Eigenvalue problem, 100  
 Eigenvectors, 100  
 Elements, 162  
 Emergency line, 239  
 End node, 73  
 End vertex, 161  
 Engelstadt, D. G., 17, 21, 54, 55, 60, 69  
 Entering basic variable, 112  
 Equations  
   consistent, 250  
   homogeneous, 250  
   linear, 250  
   linearization, 253  
 Error, control, 85  
 Euler, Leonhard, 149  
 Evaluation function, 209, 286

## F

Farber, D. J., 21  
 Feedback, 286  
   path, 6  
 Feller, W., 244  
 Fiacco, A. V., 148  
 FIBONA, subroutine, 294  
 Fibonacci numbers, 282  
 Fibonacci search, 109, 125, 130, 138, 281  
 Final value theorem, 45  
 Florian, M., 222

Flowgraph algebra, 10  
 FMALFA, subroutine, 294  
 Forbidden edge, 194, 286  
 Ford, L. K., 295  
 Forest, 165, 286  
 FORTRAN, 41  
 Forward edge, 29, 286  
 Forward path, 6, 286  
 Free edge, 194, 286  
 Freeman, R. J., 221  
 Frequency response analysis, 45  
 FREV, row, 43  
 Friedman, D. P., 21  
 Fulkerson, D. R., 295  
 Fundamental circuit, 162, 185  
 Fundamental cut-sets, 171  
   equations, 186

## G

*G* level FORTRAN, 41  
*G* vector, 41, 43  
 Gain, 2, 286  
   sensitivity, 63  
 Gandhi, S., 223  
 Garfinkel, R. S., 222  
 Gauss-Jordan, 12, 110, 274  
 Gaussian elimination, 250  
 Goeffrion, A. M., 222  
 Gradient, 123  
 Graph,  
   acyclic, 6  
   bipartite, 171, 285  
   connected, 286  
   cyclic, 6, 286  
   directed, 1, 286  
   undirected, 161, 288  
 GRASPE, subroutine, 10, 21  
 Greenberg, H., 222  
 Grens, E. A., 108  
 Griswold, R. E., 21  
 Gutterman, M. M., 222

## H

Hall, P., 173, 174  
 Hamiltonian cycle, 192, 286  
 Hammer, P. L., 221  
 Harary, F., 159, 172, 295  
 Hart, T. P., 21

Hashimoto, I., 107  
 Haskins, D. E., 17, 21  
 Heat  
   exchanger network, 78  
   exchanger problem, 131  
   transfer coefficient, 82  
 Hegerich, R., 222  
 Held, M., 221  
 Henry constant, 72  
 Heuristical, 286  
 Hillier, F. S., 148, 174, 203, 221, 222, 275  
 Himmelblau, D. M., 108  
 Hot standby module, 226  
 Howard, R., 188  
 Hyperplane, 127

## I

IBM 360/44, 43  
 Identity matrix, 245  
 Implicit enumeration, 189  
 Imposed edge, 194, 286  
 Incidence  
   equations, 186  
   matrix, 176, 182, 287  
 Index, 12, 287  
 Indicator, 287  
 Inequality constraint, 110  
 Infeasible solution, 120  
 Initial node, 3  
 Inoue, K., 223  
 Input node, 3, 161, 287  
 Inputs, 2

## J

Junction point, 287

## K

Kalamazoo, 153  
 Karel, G., 221  
 Karp, R. M., 221  
 Kaufman, A., 222, 295  
 Kesavian, H. K., 188  
 Kirchoff circuit, 163  
 Knapsack problem, 221  
 Koenig, H. E., 188  
 Königsberg bridge, 149  
 Koppel, L. B., 108  
 Kurosu, K., 244

## L

Land, A. H., 206, 221  
 Laplace operator, 2  
 Lawler, E. L., 167, 174, 221  
 Lead-lag controller, 84  
 Leakage transmittance, 57  
 Leaving basic variable, 112  
 Ledet, W. P., 108, 174  
 Lee, K. F., 174, 222  
 Lee, W., 108  
 Level, 287  
 Levin, M. I., 21  
 Lieberman, G. J., 148, 174, 203, 221, 275  
 Lin, P. M., 21  
 Linear equations, 230  
 Linear inequalities, 269  
 Linear programming, 24, 109, 269  
 Link, 162, 287  
 LISP, subroutine, 10, 21  
 List processing, 33  
 Little, J. D. C., 189, 221  
 Llewellyn, R. W., 24, 43  
 Lockett, J. A., 174, 295  
 Logarithmic sensitivity, 56  
 Log-mean temperature, 138, 147  
 Logic tree, 163, 287  
 Loop, 6, 161  
   gains, 16, 35, 287  
 LOOPS, subroutine, 42, 81, 88, 105, 289, 293  
 Lorens, C. S., 11, 295

## M

MAIN program, 41, 290, 293  
 MANS LP, program, 118, 121  
 Marriage problem, 172  
 Mason, S. J., 21  
 Mason, T., 222  
 MASON 1, subroutine, 290  
 MASON 2, subroutine, 290  
 MASON S, subroutine, 293  
 Mason's rule, 15, 257  
 Masso, A. H., 174, 222  
 Matrix  
   addition, 246

- adjacent, 176, 179, 266, 285
  - adjoint, 100, 249
  - associated, 176, 285
  - augmented, 250
  - Boolean, 176
  - circuit, 176, 184, 286
  - cofactors, 249
  - connection 14, 176, 286
  - cut-set, 176, 185, 286
  - cycle, 286
  - diagonal, 245
  - identity, 245
  - incidence, 176, 182, 287
  - inverse, 246, 247
  - MARKOV, 176, 181, 287
  - minor, 249
  - partitioning, 247
  - relation, 176
  - singular, 246
  - skew-symmetric, 246
  - square, 245
  - structural, 176
  - symmetric, 246
  - theory, 245
  - transition, 176
  - transpose, 246
  - Maximal cycle, 6, 287
  - Maxwell, L. M., 295
  - McCarthy, J., 21
  - McCormick, G. P., 148
  - McFatter, W. E., 244
  - McMahon, G., 222
  - Mitra, S. K., 69
  - Module,
    - multi-unit, 224
    - single unit, 224
  - Modulo 2 algebra, 175, 265
  - Moodie, C. L., 222
  - Moore, E. F., 25, 43
  - Morello, V. S., 244
  - MTBF (mean time between failures), 242-243
  - Multilevel technique, 78
  - Multiplicity, 64
  - Munoz, R. M., 69
  - Murayama, Y., 244
  - Murill, P. W., 188
  - Murty, D. W., 221
- N**
- NAND structures, 221
  - NEDGES, row, 43
  - NEDGS 1, row, 43
  - Nelson, A. C., 244
  - Nemhauser, C. L., 221, 222
  - Network, 287
  - Network functions, 56
  - Nilsson, N. J., 216, 221
  - Node
    - closed, 194, 286
    - current, 191, 286
    - degree, 287
    - dummy, 286
    - end, 73
    - initial, 3
    - input, 3, 161, 287
  - Nonbasic variable, 110
  - Nondirected graphs, 176
  - Nonlinear programming, 109
  - Norman, R. L., 108, 295
  - Nullity, 167, 287
  - Null-return difference, 58
  - Nyquist plot, 51
  - Nyquist stability, 51
- O**
- Offer, 209
  - OGAIN, subroutine, 42, 74, 81, 88, 290
  - Ohno, H., 107
  - One-connection, 278
  - Open loop, 49
  - Open-loop transfer function, 89
  - Open path, 287
  - Ore, O., 295
  - Origin node, 73, 161
  - Orthogonal, 127
  - OUTPUT, 88
  - Output
    - node, 287
    - sets, 172
    - variable, 287
- P**
- P array, 41, 43
  - Padé approximation, 86

Path, 6, 286  
   acyclic, 161  
   closed, 6  
   direct, 6  
   feedback, 6  
   finding algorithm, 229  
   forward, 6, 286  
   gain, 6, 287  
   inverse, 6  
   transmittance, 6  
   union, 227  
   unnecessary, 36  
 PATHP, subroutine, 290  
 PATHS, subroutine, 42, 74, 88, 105, 289  
 Pending nodes, 189, 287  
 Permutations, 25  
 Phase angle, 47  
 Picnic problem, 172  
 Pike, R. W., 188  
 PLENTH, subroutine, 291  
 Pointer, 37  
 Pole sensitivity, 64  
 Polovko, A. M., 244  
 Postoptimal analysis, 121  
 Pressure detector, 239  
 Primary variables, 184  
 PRINTR, subroutine, 42, 290  
 Process reliability, 223  
 Product rule, 10  
 Pruning, 207, 287  
 Pseudo-Boolean programming, 205, 287

## R

Radanovic, L., 69  
 Rank, 170, 287  
 Rankings, 149  
 Rate constants, 61  
 Reachability matrix, 287  
 Recycle, 60  
   calculations, 102  
 Redei, 149  
 Redundancy,  
   active, 224, 225  
   standby, 224  
 Reed, M. B., 163, 171, 295  
 Relation matrix, 176  
 RELCOMP, subroutine, 243  
 Reliability analysis, 223

Reliability graph, 224  
 Relief valve, 239  
 Residual graph, 6, 287  
 Resonant peak, 50  
 Return difference, 58  
 Reverse edge, 29  
 Reverse path, 6  
 Robert, J., 12, 13, 295  
 Robichaud, L. P. A., 12, 13, 295  
 Robillard, P., 222  
 Root, 287  
 Root locus diagram, 51  
 ROOTS, subroutine, 100, 102, 290  
 Roy, B., 221  
 Rudd, D. F., 108, 174, 221, 222

## S

Saatý, T. L., 295  
 Sargent, R. W. H., 108  
 Scoring function, 287  
 Search trees, 187  
 Secondary variables, 184  
 Sedlar, M., 67, 69  
 Self-loop, 12, 287  
 SENSE, subroutine, 92, 290  
 Sensitivities, 56  
 Sensitivity analysis, 53, 122, 227  
   logarithmic, 56  
   points method, 67  
 Seshu, S., 163, 171, 295  
 Shaftel, T., 222  
 Shooman, M. L., 244  
 Short circuit, 62  
 Signal flow graph, 2, 287  
   construction, 7  
   reduction, 9  
 Sirola, J. S., 221  
 Simplex algorithm, 111, 273  
 Sink, 3, 287  
 Slack variables, 110, 270  
 Sliepcevich, C. M., 17, 21  
 SLIP, subroutine, 10  
 SNOBOL, 10, 21  
 Solutions  
   basic feasible, 272  
   optimal, 272  
 Source, 3, 287  
 SPAN, subroutine, 162

Square matrix, 245  
 Standby redundancy, 224  
 State  
   enumeration algorithm, 231  
   equation, 183  
   model, 183  
   variable, 183  
 Status vector, 36  
 Steady state gain, 89  
 Step response, 50  
 Steward, D. V. 222  
 Stirred tank reactors, 60  
 Stream frequency, 103  
 Structural analysis, 60, 83  
   matrix, 176  
   sensitivity method, 65  
 Strum, R. D., 20, 21  
 Subgraph, 278  
 Substitution rule, 47  
 Subtour, 287  
   elimination, 204

## T

Takamatsu, T., 83, 107, 131, 133, 148  
 Tank system, 54  
 Taylor series, 84, 253  
 Temperature driving force, 78  
 Tennis rankings, 150  
 Texas Father and Son, 152, 158  
 Terano, T., 244  
 Tergny, J., 221  
 Terminal graph, 183, 287  
 Terminal node, 192  
 Terminal variables, 183, 287  
 Terminal vertex, 3  
 Through variables, 287  
 Tokada, V., 188  
 Tomovic, R., 67, 69  
 Tonomura, T., 109, 148  
 Toray industries, 109  
 Torn streams, 103  
 Tour, 192, 288  
   building algorithm, 204  
 Transfer functions, 68, 288  
 Transition matrix, 176, 181, 288  
 Transmission function, 288  
 Transmission rule, 10

Transmittance, 2, 15, 288  
 Transportation problem, 24, 105  
 Traveling salesman problem, 192  
 Tree, 161, 288  
   search, 189  
 Trepant, P., 222

## U

Ufford, P. S., 224  
 Undirected graphs, 161, 288  
 Unit step change, 96  
 Unnecessary paths, 36  
 Upadhye, R. S., 108

## V

VALUE, function, 294  
 Valve, servo, 239  
 Varied branch transmittance, 66  
 Vectors  
   control, 255  
   direction, 121  
   dot product, 248  
   independence, 248  
   inner product, 248  
   loop, 37  
   orthogonality, 248  
   rank, 248  
   state, 255  
 Vertex  
   edge, 286  
   matrix, 288

## W

Ward, J. R., 20, 21  
 Warm standby module, 225  
 Washi, P. N., 222  
 Weinblatt, H., 108  
 Westerberg, A., 108  
 Wilde, D. J., 282  
 Wood, D. E., 167, 174, 222

## Z

Zadeh, L. A., 257, 263  
 Zero coefficient, 119  
 Zero sensitivity, 64  
 Zoutenijk, G., 109, 125, 131, 148

4  
 B 5  
 C 6  
 D 7  
 E 8  
 F 9  
 G 0  
 H 1  
 I 2  
 J 3